

THESIS
NASA
11-67
8525

P-176

LEARNING FUZZY LOGIC CONTROL SYSTEM

by

Leung Kam Lung

B. S., Metropolitan State College at Denver, 1991

N96-19289

Unclass

G3/67 0101136

A thesis submitted to the

Faculty of the Graduate School of the

University of Colorado at Denver

in partial fulfillment

of the requirements for the degree of

Master of Science

Electrical Engineering

1994

(NASA-CR-200241) LEARNING FUZZY
LOGIC CONTROL SYSTEM M.S. Thesis
(Colorado Univ.) 176 p

This thesis for the Master of Science

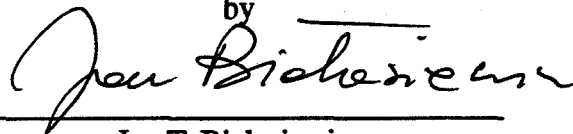
degree by

Leung Kam Lung

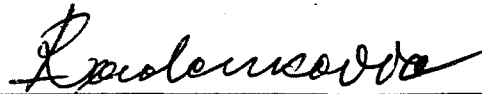
has been approved for the

Graduate School

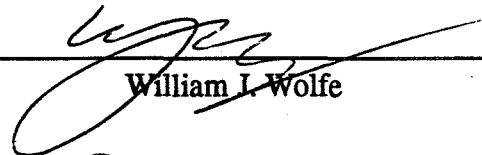
by



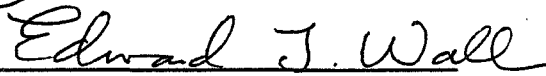
Jan T. Bialasiewicz



Miloje Radenkovic



William I. Wolfe



Edward T. Wall

July 14, 1994
Date

Leung Kam Lung (M.S., Electrical Engineering)

Learning Fuzzy Logic Control System

Thesis directed by Associate Professor Jan T. Bialasiewicz

ABSTRACT

The performance of the Learning Fuzzy Logic Control System (LFLCS), developed in this thesis, has been evaluated. The Learning Fuzzy Logic Controller (LFLC) learns to control the motor by learning the set of teaching values that are generated by a classical PI controller. It is assumed that the classical PI controller is tuned to minimize the error of a position control system of the D.C. motor. The Learning Fuzzy Logic Controller developed in this thesis is a multi-input single-output network. Training of the Learning Fuzzy Logic Controller is implemented off-line. Upon completion of the training process (using Supervised Learning, and Unsupervised Learning), the LFLC replaces the classical PI controller. In this thesis, a closed loop position control system of a D.C. motor using the LFLC is implemented. The primary focus is on the learning capabilities of the Learning Fuzzy Logic Controller. The learning includes symbolic representation of the Input Linguistic Nodes set and Output Linguistic Nodes set. In addition, we investigate the knowledge-based representation for the network. As part of the design process, we implement a digital computer simulation of the LFLCS. The computer simulation program is written in "C" computer language, and it is implemented in DOS platform. The LFLCS, designed in this

thesis, has been developed on a IBM compatible 486-DX2 66 computer. First, the performance of the Learning Fuzzy Logic Controller is evaluated by comparing the angular shaft position of the D. C motor controlled by a conventional PI controller and that controlled by the LFLC. Second, the symbolic representation of the LFLC and the knowledge-based representation for the network are investigated by observing the parameters of the Fuzzy Logic membership functions and the links at each layer of the LFLC. While there are some limitations of application with this approach, the result of the simulation shows that the LFLC is able to control the angular shaft position of the D.C.motor. Furthermore, the LFLC has better performance in rise time, settling time and steady state error than to the conventional PI controller.

This abstract accurately represents the content of the candidate's thesis. I recommend its publication.

Signed Jan Bialasiewicz
Jan T. Bialasiewicz

ACKNOWLEDGEMENTS

This study was supported by a NASA grant #NAG-1-1444. I have also received much support and encouragement from many people.

Especially, I would like to express my sincere gratitude to my advisor, Professor Jan T. Bialasiewicz, and Professor William J. Wolfe for their support and guidance during the preparation of this thesis. I also would like to thank Professor Miloje Radenkovic and Professor Adward T. Wall for their encouragement and also for serving on my committee.

Finally, this thesis is dedicated to my parents for their love and caring. Heavenly father, thank you for helping me to finish this thesis.

CONTENTS

Chapter 1

1.1	Introduction.....	1
1.2	Previous Research.....	2
1.3	Structure of the LFLC: An Overview.....	5

Chapter 2

2.1	Normalization	8
-----	---------------------	---

Chapter 3

3.1	Network Structure.....	11
3.2	Input Linguistic Nodes Layer	11
3.3	Input Term Nodes Layer	11
3.4	Rule Nodes Layer	13
3.5	Output Term Nodes Layer	15
3.5.1	Down-Up Mode	15
3.5.2	Up-Down Mode	18
3.6	Output Linguistic Nodes Layer.....	18

Chapter 4

4.1	Learning Phase.....	20
4.2	Unsupervised Learning	20
4.3	Supervised Learning	22

Chapter 5

5.1	Symbolic Representation and Performance Evaluation of the LFLC..	27
-----	--	----

5.2	Low Level Symbolic Representation of the LFLC.....	27
5.3	High Level Representation of the LFLC	35
5.4	Performance evaluation	38
Chapter 6		
6.1	Summary	41
Appendix A		42
Appendix B.....		45
REFERENCES.....		166

FIGURES

Figure

1. Learning Fuzzy Logic Control System.....	5
2. Learning Fuzzy Logic Controller	10
3. Initial Input Term Node membership for x_1	28
4. Initial Input Term Node membership for x_2	29
5. Final Input Term Node membership for x_1	30
6. Final Input Term Node membership for x_2	31
7. Convergence of the Input Term Node memberships for input x_1	32
8. Convergence of the standard deviations of the Input Term Node membership for x_1	32
9. Convergence of the means of the Input Term Node memberships for input x_2	33
10. Convergence of the standard deviations of the Input Term Node memberships for input x_2	33
11. Initial Output Term Node membership	37
12. Final Output Term Node membership	37
13. Training Data	38
14. LFLCS with PI Fuzzy Logic Controller	39
15. Step response of the system with classical and fuzzy logic PI controller	40

Chapter 1

1.1 Introduction

In the design of Analog and Digital Control systems, a dynamic representation of the system is required. In design engineering, this information is usually not known *a priori*. Furthermore, in order to minimize the errors between the output and the input, modern control theory requires that an accurate model of the system be available. These requirements limit the application of modern control theory in many areas. It is important that controllers be developed that do not have such stringent requirements. The goal of this research is then to explore an alternative controller and to evaluate its performance. In this thesis, the concepts of Fuzzy Logic Control and Neural Network Learning are combined to design a controller for an unknown plant. The developed system is referred to as a Learning Fuzzy Logic Control System (LFLCS).

Whereas Unsupervised Learning Neural Network is used to set up the initial structure for the network controller, Supervised Learning Neural Network is used to adjust parameters of the network controller to minimize the output error. This research addresses the learning capability of the Learning Fuzzy Logic Controller (LFLC), and its knowledge representation in symbolic terms. A complete system diagram is shown in Fig.1. The LFLCS requires that the input signals and the teaching signal be available for training, and that a good logical structure be set up before the training takes place. The LFLC is rather different from a conventional controller; this difference is explained and illustrated.

1.2 Previous Research

Modern control theory has proved to be very useful in areas where systems are well defined either deterministically or stochastically. However, many control systems involve human-judgement interaction as part of the control system. Human involvement often provides an adequate controller because the mind of an operator usually provides a model of the process which is just accurate enough to carry out the task at hand. On the contrary an automatic controller has no way of observing the essential features of a particular process. A human is usually capable of learning through experience, which decreases the need for a precise model of the system. Thus, modelling the human decision making process is essential in control system design. The knowledge of the control process in the human mind is captured in the fuzzy system design approach.

In 1965, Zadeh [31] introduced and developed the concept of fuzzy set theory. Since its introduction, fuzzy logic has been successfully applied in many control system applications; for example, see references [1], [29], [6], [17], [16], [20] for some good illustrations. An excellent overview of fuzzy logic applications in control engineering is given by Langari and Berenji [15]. In reference [4], Chuen-Tsai Sun implemented the fuzzy IF-THEN rule base to identify the structure and the parameters of a network such that a desired input-output mapping is achieved. However, a fuzzy logic controller requires that the control strategy be obtained as a fuzzy set term. This limits somewhat the usefulness of the concept of a fuzzy controller.

The Self-Organizing (unsupervised learning) approach was presented by Zhang and Edmunds [32], and Linkens and Hasnain [20]. They proposed a fuzzy logic con-

troller that is able to cluster (self-organize) the input data without any prior knowledge of the data, and the network automatically sets up the parameters for each membership of the network. Furthermore, this network has a learning algorithm and is capable of generating and modifying control rules based on an evaluation of the system performance. The generation and modification of the control rules is achieved by assigning a credit or reward value (weight) to the individual control action that makes a major contribution to the current performance. This is an excellent control strategy for a system when the operator control strategy is not available.

The concept of machine learning was introduced many years ago in an effort to achieve human like performance in the fields of speech and image recognition for handicapped people. An extensive research effort to simulate the process of intelligent human learning using an Artificial Neural Net using the neural network is a major effort in the field of computer sciences [2], [9], [11], [12], [24], [30], [26]. Also, Self-Organized (unsupervised learning) is one of the learning methods which is used in speech recognition. Supervised learning is used in many fields where teaching data are obtainable from an applicable sources [21], [22], [10], [27], [8], [25].

Regardless of the name these models attempt to achieve a good performance via a dense interconnection of simple computational elements. In this respect, an artificial neural net structure is based on present measurements of biological systems. Neural net models have the greatest potential in application such as speech, image recognition, and control systems where many of the hypotheses require a parallel, high computational rate, and the better systems result from observing inadequate human performance. It is hoped that the potential benefits of neural nets will extend beyond

the high computation rates provided by present massive parallel system. Neural nets typically provide a greater degree of robustness or fault tolerance than the von Neumann sequential computers of today because there are many more processing nodes possible, each with primarily local connections. The presence of a few imperfect nodes or links thus need not impair the overall performance significantly. Most neural net algorithms also adapt connection weights in order to improve the performance based on current results. Work on artificial neural net models has a long history. Development of detailed mathematical models began more than 45 years ago with the work of McCulloch and Pitts [23]. Lin and Lee presented in [18] a two-phase learning fuzzy logic network which consisted of both unsupervised learning and supervised learning, and in [19] they developed a reinforcement neural network-based fuzzy logic control system.

In this thesis, following the approach of Lin and Lee [18], the Learning Fuzzy Logic Control System (LFLCS) is proposed in which the learning capabilities of neural networks are utilized. The system learns by adjusting the parameters of the neural network using training data. The learning schemes of a Learning Fuzzy Logic Controller (LFLC) combine both the unsupervised (self-organized) and the supervised learning.

The neural network structure, implementing the LFLC, is given in Fig.2. This network has one output and two inputs. The input signals x_1 , and x_2 to the LFLC are the feedback signal and the error signal of the control system, respectively, as shown in Fig.1. The teaching pattern used by the LFLC is the control signal generated by a conventional controller.

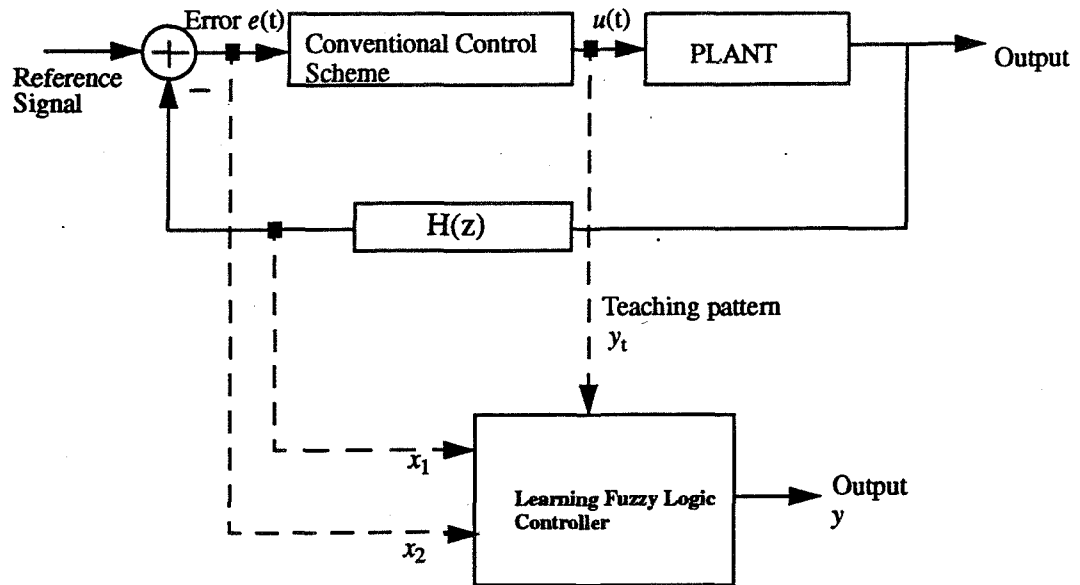


Fig.1. Learning Fuzzy Logic Control System

1.3 Structure of the LFLC: An Overview

A Neural Network (or connectionist network) is a highly parallel connected network that attempts to model the learning ability of the human brain. The intelligence of the network is represented by the weights that connect nodes at one layer to the nodes of the next layer of the network. Fuzzy Logic Control is a knowledge-based control strategy. This strategy can be employed given a sufficiently accurate control law which is not unreasonably complex. The Neural Network learns by the tuning of system parameters using training data. The learning schemes of the LFLC combine both the unsupervised (self-organized) learning and the supervised learning. A layout of a simple network is found in Fig. 2. This particular network has two inputs and one output, i.e., it belongs to a Multiple Input Single Output (MISO) class of networks. As

shown in Fig.1, the input signal (x_1) to the LFLC is the feedback signal of the control system, and the input signal (x_2) to the LFLC is the error signal of the control system. The LFLC learns to recognize a set of data. This set of data is called the teaching pattern which is generated by the digital Proportional-Integral (PI) controller in Fig. 1.

The Neural Network is used as a learning mechanism and the Fuzzy Logic Control algorithm is actually controlling the plant. The LFLC is a two phase learning network. The first phase is an unsupervised learning phase, and the second is supervised learning phase. The LFLC has a total of five layers. The first layer is the Input Linguistic Nodes layer which in this particular implementation contains two nodes. These nodes represent the input data sets as symbolic terms. A set of five nodes is set up in the second layer for each Input Linguistic Node. This is the Input Term Nodes layer. The purpose of the Input Term Nodes is to categorize the input data into linguistic terms. The third layer is the Rule Nodes layer. Each rule node in the third layer represents a rule of controlling the plant. The number of rule nodes in this layer is equal to the product of the numbers of nodes in each set of the Input Term Nodes. Therefore, there are 25 rule nodes in our implementation. The fourth layer is the Output Term Node layer that contains seven nodes. The purpose of the Output Term Node is to categorize in linguistic terms the consequences of the fired rules. The fifth layer is the Output Linguistic Nodes layer that in the case of the application considered contains one output node. However, a second node is used to train the Output Linguistic Node in the unsupervised learning phase. The Output Linguistic Node constitutes the output of the LFLC. The data are randomly presented to the LFLC in the learning phases. How-

ever, before the data are presented to the network, they go through a normalization process, which is explained in detail in the next section.

Chapter 2

2.1 Normalization

In the normalization process, the data of the error signal (x_2) and the feedback signal (x_1) are mapped to the range of $[-1,1]$. The data of the teaching pattern (y_i) are mapped to the range of $[0,1]$. These mappings are accomplished by the data normalization using the following:

$$\bar{x}_i = \frac{x_i[j]}{\max(|\max x_i[j]|, |\min x_i[j]|)}, \quad i = 1, 2 \quad j = 1, 2, \dots, n \quad (1)$$

$$y_i = 0.5 + \frac{y_i[j]}{2 \cdot \max(|\max y_i|, |\min y_i|)}, \quad j = 1, 2, \dots, n \quad (2)$$

where n denotes the number of data points.

Since $\{x_i[j]\}$ contains all data of the linguistic node (x_i) that are going to be normalized by Eq.(1), the $\{y_i[j]\}$ contains all data of the linguistic node (y_i) that will going be normalized by Eq(2). The $\max x_i$ becomes the maximum number of the set $\{x_i[j]\}$, and $\min x_i$ the minimum number of the set $\{x_i[j]\}$. In the same manner the $\max y_i$ is the maximum number of the set $\{y_i[j]\}$, and $\min y_i$ the minimum number of the set $\{y_i[j]\}$. The $\bar{x}_i$ is the result of the normalization of the data of the linguistic node (x_i), and y_i is the result of the normalization of the data of the teaching patterns y_i . This normalization ensures that the input data and the teaching data are mapped to the ranges $[-1, 1]$ and $[0, 1]$ respectively. All negative values of the original teaching data

are mapped to $[0, 0.5)$, and all positive values of the original teaching data are mapped to $(0.5, 1]$. Similarly, all negative values of the original input data are mapped to $[-1, 0)$, and all positive values of the original input data are mapped to $(0, 1]$.

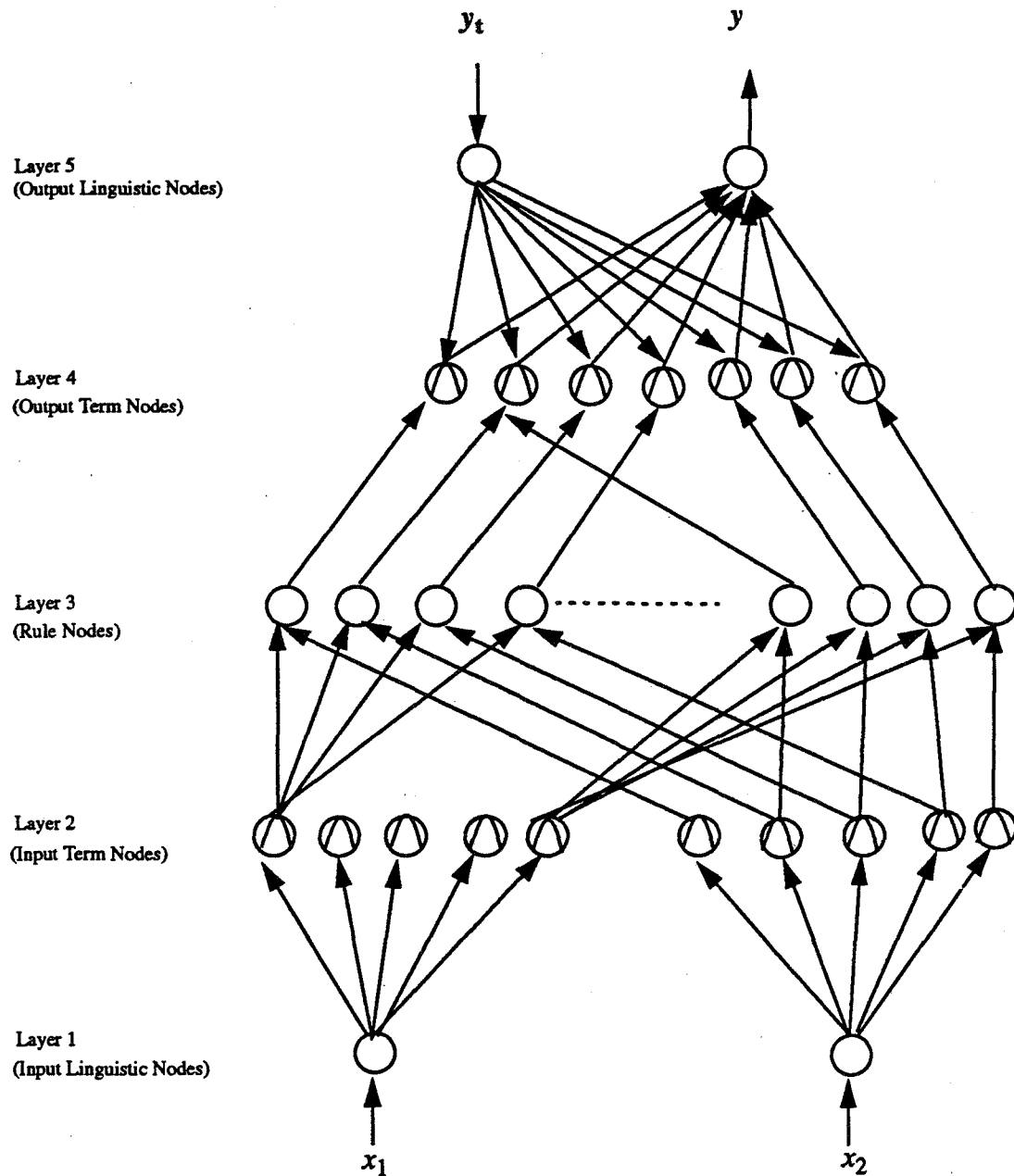


Fig.2. Learning Fuzzy Logic Controller

Chapter 3

3.1 Network Structure

The LFLC has a total of five layers. These are the following:

First layer or Input Linguistic Nodes Layer

Second layer or Input Term Nodes Layer

Third layer or Rule Nodes Layer

Fourth layer or Output Term Nodes layer

Fifth layer or Output Linguistic Nodes Layer.

Each of these layers is connected by the link between them. The purpose and detail of each layer will be explained in the following sections.

3.2 Input Linguistic Nodes Layer

The purpose of this layer is to propagate the input data to the Input Term Nodes layer; therefore, each input linguistic node is connected to a set of Input Term Nodes in the next layer. The output value of the input linguistic node is the same as the input value, and is propagated to its own Input Term Nodes set in the next layer. The link (w) from first layer to the second layer is unity.

3.3 Input Term Nodes Layer

The purpose of the Input Term Node is to represent the value of the Input Linguistic Node in linguistic terms. For example, the linguistic term for an Input Term Node can be Negative Large (NL), Negative Medium (NM), Negative Small (NS), Zero (ZN), Positive Small (PS), Positive Medium (PM), or Positive Large (PL).

Two sets of Input Term Nodes are set up for the LFLC as shown in Fig. 2. Five nodes are set up for each Input Term Nodes set; thus, for this particular network, a total of 10 nodes are in the Input Term Nodes layer. The input value to each node in the Input Term Nodes set number 0 is equal to the product of the output value of the Input Linguistic Node number 0 and the link (w) that connects to the Input Term node. Each node of the Input Term Nodes set has a membership function. This membership function can be based on any activation function (a); however, a Gaussian activation function is used in the network considered. This Gaussian activation function is defined by (3) and (4) below. It is used for all memberships in layer 2,

$$a = e^f \quad (3)$$

$$f = M_{u_i}^j(m_{ij}, \delta_{ij}) = \frac{-\left(u_i^2 - m_{ij}\right)^2}{\delta_{ij}^2} \quad (4)$$

where f is the memberships function $M_{u_i}^j(m_{ij}, \delta_{ij})$. The parameters m_{ij} and δ_{ij} are the center (the mean) and the width (the variance) of the Gaussian function respectively, and u_i^2 is the input data to the Gaussian function with the superscript 2 used as a reference to the second layer. The subscript i is the index of the Input Term Nodes set, and the subscript j is the index of the nodes within each Input Term Nodes set. The links (w) of this layer are fully connected and they are equal to unity (See Fig. 1.). For our particular structure shown in Fig. 2, $i = 1, 2$ and $j = 1, 2, 3, 4, 5$. We have $u_1^2 = x_1$ and $u_2^2 = x_2$.

3.4 Rule Nodes Layer

This layer contains a set of fuzzy logic rules R_i . For this thesis, a MISO system is used. A linguistic variable (x) in a universe of discourse U is characterized by two sets: $T(x) = \{T_x^1, T_x^2, \dots, T_x^k\}$ and $M(x) = \{M_x^1, M_x^2, \dots, M_x^k\}$. The $T(x)$ is the term set of x which is the set of names in linguistic terms of the values of x with each value T_x^i being a fuzzy number with membership function M_x^i defined on U . Thus $M(x)$ is a semantic rule for associating each value with its meaning.

For example, if x indicates voltage, then $T(x)$ may be in the set of $\{NL, NS, ZN, PS, PL\}$. In this thesis, five Input Term Nodes are set up for each Input Linguistic Node, and seven Output Term Nodes are set up for each Output in this network, i.e., $T(y) = \{T_y^1, T_y^2, \dots, T_y^7\} = \{NL, NM, NS, ZN, PS, PM, PL\}$. The fuzzy logic rules for the LFLC are stated as follows:

$$R_i \equiv \text{IF } x_1 \text{ is } NL \text{ and } x_2 \text{ is } ZN, \text{ THEN the consequence is } PM$$

where $i = 1, 2, \dots, 25$. Thus, a total of 25 rule nodes are in this layer. The input to each rule node u_j^3 comes from one possible combination of the outputs of the Input Term Nodes set $o_{i,k}^2$, where $j = 1, 2, \dots, 25$ and $i = k = 1, 2, \dots, 5$. Let $o_{1,1}^2$ and $o_{2,1}^2$ be the possible inputs to the rule node R_1 ; however, only the smallest value of $o_{i,k}^2$ becomes the input to the rule node R_1 . Furthermore, the links (w) of this layer are unity. If there are two rules:

$$R_1 \equiv \text{IF } x_1 \text{ is } NL \text{ and } x_2 \text{ is } ZN, \text{ THEN the consequence is } PM \quad (5)$$

$$R_2 \equiv \text{IF } x_1 \text{ is NL and } x_2 \text{ is ZN, THEN the consequence is PS} \quad (6)$$

then the firing strengths of R_1 and R_2 are denoted as α_1 and α_2 , respectively. For example, α_1 is given by the following equality:

$$\alpha_1 = M_{x_1}^{NL}(x_1) \wedge M_{x_2}^{ZN}(x_2) \quad (7)$$

where $\wedge$ is the fuzzy logic AND operation. The intersection is used as the fuzzy logic AND operator. Thus, the AND operator is realized by the following equation:

$$\alpha_i = M_{x_1}^q(x_1) \wedge M_{x_2}^q(x_2) = \begin{cases} \min(M_{x_1}^q(x_1), M_{x_2}^q(x_2)) \\ \text{or} \\ M_{x_1}^q(x_1) M_{x_2}^q(x_2) \end{cases} \quad (8)$$

where q is one of the $\{NL, NS, ZN, PS, PL\}$ and $i = 1, 2, \dots, 25$. The nodes in this layer form a fuzzy rule base. The connectionist inference engine is constructed by combining the functions of this layer and the functions of layer 4 [28], [7], [5]. Hence, the rule matching process is avoided [18]. The precondition matching of fuzzy logic rules is accomplished by the method of linking layers. Each rule node in this layer performs the fuzzy AND operation; thus the activation function (a) for the rule node is the minimum of all its inputs.

$$a = f \quad (9)$$

$$f = \min(u_1^3, u_2^3, \dots, u_p^3) \quad (10)$$

and for the network analyzed $p = 2$.

The connections of this layer are given as: a node in each of the Input Term Nodes set is connected to a rule node with a constraint such that no two nodes are from the same Input Term Nodes set are connected.

3.5 Output Term Nodes Layer

This a layer has two operation modes. In the first phase of the training, the nodes of this layer operate in an up-down transmission mode. Whereas in the second phase of the training, these nodes operate in a down-up transmission mode. Upon completion of the learning process, the set $M_y = \{M_y^1, M_y^2, \dots, M_y^7\}$ of the membership functions of the Output Term Nodes set are found. An Output Term Node number j may be excited (as a result of the learning process in which the structure of connections between the Rule Nodes layer and the Output Term Nodes layer are established) by a few or none of output signals of the Rule Nodes.

3.5.1 Down-Up Mode

In this mode, the node performs the fuzzy logic OR operation. To illustrate this concept, let us assume that (as defined by the structure of connections) an Output Term Node is excited by the Rule Nodes number $i_1, i_2, \dots, i_p$ that is described in section 3.4. In our particular structure, p can be any number in the set $\{1, 2, \dots, 25\}$ at the Rule Node layer. Then, the membership function associated with the Output Term Node number j can be defined as follows:

$$\hat{M}_y^{i,j}(s_{i_k}) = \alpha_{i_k} \wedge M_y^i(s_{i_k}) = \begin{cases} \min(\alpha_{i_k}, M_y^i(s_{i_k})) \\ \text{or} \\ \alpha_{i_k} M_y^i(s_{i_k}) \end{cases} \quad (11)$$

where $i = 1, 2, \dots, p$

$$M_y^j(s_{i_k}) = \exp\left(\frac{-(s_{i_k} - m_y)^2}{\delta_y^2}\right) \quad (12)$$

where s_{i_k} is the output of the rule node number i_k , m_y and δ_y are the mean and the variance of the membership y , respectively. In the case considered, we have a set of membership functions $M_y = \{M_y^1, M_y^2, \dots, M_y^r\}$. Combining (8) and (11), we obtain the output decision:

$$\hat{M}_y^j = \hat{M}_y^{i_1j}(s_{i_1}) \vee \hat{M}_y^{i_2j}(s_{i_2}) \vee \dots \vee \hat{M}_y^{i_rj}(s_{i_r}) \quad (13)$$

where $\vee$ is the fuzzy logic OR operation which performs the UNION of a given set of memberships. Thus, the output decision for an Output Term Node number j can be written as follows:

$$\hat{M}_y^j = \begin{cases} \max(\hat{M}_y^{i_1j}(s_{i_1}), \hat{M}_y^{i_2j}(s_{i_2}), \dots, \hat{M}_y^{i_rj}(s_{i_r})) \\ \text{or} \\ \min(1, \sum_{k=1}^r \hat{M}_y^{i_kj}(s_{i_k})) \end{cases} \quad (14)$$

As a result, the nodes in this mode perform the fuzzy logic OR operation to integrate the fired rule nodes, which have the same consequence; that is, they are connected to the same Output Term Node j . This can be also expressed as

$$a_j = \min(1, f_j) \quad (15)$$

where

$$f_j = \sum_{k=1}^p \hat{M}_y^{i_k j}(s_{i_k}) \quad (16)$$

Each of the 25 Rule Nodes can be potentially connected to each of the seven Output Term Nodes as can be seen from Table 1 showing connection weights. Those weights, either 0 or 1, are established in the learning phase and tell us which i_k 's are involved in equation (13) for a given node number j .

TABLE 1. The links (weights) of the Output Term Node Layer
Rule number 1 - 25

M1	0	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	1	1	1	1	1	1	1	1
M2	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
M3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
M4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
M5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
M6	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
M7	0	0	1	1	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

3.5.2 Up-Down Mode

The purpose of this mode is to find the initial means and the variance of the Output Term Nodes. This mode is used in the first (unsupervised) learning phase. In this mode, the nodes in this layer, function the same as those in the second layer, except that only one node is used to perform a membership function for the Output Linguistic Node.

3.6 Output Linguistic Nodes Layer

This layer contains the nodes performing the UPDOWN transmission and the nodes performing the DOWNUP transmission. The UPDOWN transmission nodes are used to feed the training data into the LFLC network [18]. Their activation is defined as follows:

$$a = f = y \quad (17)$$

The DOWNUP transmission nodes together with the links that are attached to the Output Linguistic Node, performs the fuzzy OR operation or in other words implements the so called defuzzifier [18]. In this research, the fuzzy OR operation is based on the center of area method, which as described in [3], gives the best result. Let s_j be the support value, i.e., a value at which the membership function, $\hat{M}_y^j(s)$, reaches the maximum value $\hat{M}_y^j(s) |_{s=s_j}$. Then, from (13) the defuzzification output is

$$y = \frac{\sum_j \hat{M}_y^j(s_j) s_j}{\sum_j \hat{M}_y^j(s_j)} \quad (18)$$

The following equations are used to simulate the center of area defuzzification method [32]:

$$f = \sum w_{ij}^s \cdot u_i^s = \sum (m_{ij}^s \delta_{ij}^s) \cdot u_i^s \quad (19)$$

$$a = \frac{f}{\sum \delta_{ij}^s \cdot u_i^s} \quad (20)$$

where the w_{ij}^s is the link between the Output Term Node number j and the Output Linguistic Node number i , it is also equal to the product of $m_{ij}^s \cdot \delta_{ij}^s = m_{ij}^4 \cdot \delta_{ij}^4$. Here m and δ are the mean and the variance of the Output Term Node, respectively. In this particular network, we only have one Output Linguistic Node, thus j is equal to 1.

Chapter 4

4.1 Learning Phase

In LFLC there are two learning phases: the first one implements the Unsupervised or self-organized learning, and the second one implements the Supervised Learning using Backpropagation. As a result of Unsupervised Learning, the structure of the LFLC is established after the links (w) and the firing strengths of the fuzzy logic rule nodes in the network are found. Furthermore, the means (m) of each membership function in the network are found by using the Self-Organized learning method which is described by the Kohonen's feature-map algorithm [14], [13].

4.2 Unsupervised Learning

The Kohonen feature-map algorithm is a self organizing method that gathers the input data into a cluster. In his algorithm, a set of weights is initially generated for each node in his network. When data is present in the network, the distances from the data to all nodes are calculated. Let $w_{i=1, j=1}$ denote the weights of the shortest distance from the data i to the node j , then the node number j is selected to be the output node of the network. Furthermore, the weights ($w_{i=1, j=1}$) and its neighbors are updated according to the following equation:

$$w_{ij}(t+1) = w_{ij}(t) + \eta(t) (x_i(t) - w_{ij}(t)) \quad (21)$$

where j belongs to the nearest neighbor, and $0 \leq i \leq N-1$. The term $\eta(t)$ is a learning rate ($0 < \eta(t) < 1$) which decreases in time.

After the mean of the Fuzzy Logic membership function is found, the corresponding variance (δ) can be found as follows:

$$\delta_i = \frac{|m_i - m_{closest}|}{\tau} \quad (22)$$

where τ is the overlap parameter. This parameter is used to control the level of overlapping between memberships in the same cluster. The range of these parameters depends on the range of the input data. However, in our network, this parameter is chosen to be 1.5.

The $m_{closest}$ is the nearest neighbor of the current mean value. Once the centers and the variances are found, the input signal and the teaching signal reach the output points at the output term nodes and the input term nodes. Next, the output of the input term node at layer two are transmitted to layer three through the initial architecture of the layer three's links. Based on the firing strengths of the rule nodes (output of the rule nodes) and the output of the output term nodes, the correct consequences of the links of each fuzzy logic rule node are found. Initially, the links (w) are fully connected. However, the competitive learning algorithm is used to update the links (weights) for each training data set. This algorithm is described by the following equation:

$$\dot{w}_{ij}(t) = o_j^4 (-w_{ij} + o_i^3) \quad (23)$$

where o_j^4, o_i^3 , are the outputs of the output term node and the input term node, respectively. w_{ij} denotes the weight of the link between the i -th rule node and the j -th output term node. A dot over w_{ij} denotes the next value of the w_{ij} .

After competitive learning, the weights of the links at layer four represent the firing strength of the rule node which is transmitting the consequence of the rule node to the term node of an output linguistic node. Furthermore, the links are chosen such that at most one link is selected and the others are eliminated. As a result, only one term node in the output term node set becomes one of the consequences of the fuzzy logic rule. The supervised learning takes place after the fuzzy logic rule is established in the unsupervised learning phase.

4.3 Supervised Learning

In the supervised learning phase, the backpropagation method is used to fine tune the parameters of the LFLC which were described above. The objective of the second phase of learning is to minimize the error function:

$$E = \frac{1}{2} (y(t) - \hat{y}(t))^2 \quad (24)$$

The error signals for layer five, layer four, and layer three, are given as:

$$\sigma^5(t) = y(t) - \hat{y}(t) \quad (25)$$

$$\sigma_i^4(t) = \sigma^5(t) \cdot \frac{m_i \sigma_i (\sum \delta_i u_i) - (\sum m_i \delta_i u_i) \delta_i}{(\sum \delta_i u_i)^2} \quad (26)$$

$$\sigma_i^3 = \sum_k \sigma_i^4 \quad (27)$$

where $y(t)$, $\hat{y}(t)$ are the desired output and the current output of the LFLC, respectively.

Backpropagation has a forward phase and a backward phase. The forward phase of the backpropagation requires that the data be presented to the network at the first

layer for training. Next, the outputs of the input term nodes are calculated, and then, transmitted to one of the output term nodes, via the firing strengths of the rule nodes. Each fuzzy logic rule was structured to be excited by the smallest output value of the initially defined input term nodes. Finally, the output of the LFLC is calculated using equation (18). After the output is found for a pair of the training data, the error signal is calculated using equation (24) and is then propagated to all of the previous layers of the LFLC. Concurrently the error signal is being used to update the set of means (m) and the set of variances (δ) of each layer in the LFLC. The means and the variances at layer five are updated (fine tuned) by equations (27) and (28), respectively.

$$m_i^5(t+1) = m_i^5(t) + \eta \cdot \sigma^5 \cdot \frac{\delta_i u_i}{\sum \delta_i u_i} \quad (28)$$

$$\delta_i^5(t+1) = \delta_i^5(t) + \eta \cdot \sigma^5 \cdot \frac{m_i u_i (\sum \delta_i u_i) - (\sum m_i \delta_i u_i) u_i}{(\sum \delta_i u_i)^2} \quad (29)$$

where the learning rate (η) is a function decreasing in value as the time progresses. Here the symbol u is the input value to the current node i and δ , m is the variance, and the mean of the current membership i , respectively. Layer four does not contain any parameters. Thus, only the error signal (σ^4) of this layer needs to be calculated, and it is then propagated to the layer three. The equation of the error signal (σ^4) is obtained from equation (25). As shows in Fig. 1., layer three also does not contain any parameters that have to be updated, hence only the error signal (σ^3) is calculated and propagated to layer two. In the layer two, the mean (m) and the variance (δ) are updated according to the following:

The general learning rule in Neural Network is

$$\Delta w \propto \left(-\frac{\partial E}{\partial w} \right) \quad (30)$$

which can be written as

$$w(t+1) = w(t) + \eta \cdot \left(-\frac{\partial E}{\partial w} \right) \quad (31)$$

where η is the learning rate and E is the error defined by eq. (24) and

$$\frac{\partial E}{\partial w} = \frac{\partial E}{\partial (net-input)} \cdot \frac{\partial (net-input)}{\partial w} = \frac{\partial E}{\partial f} \cdot \frac{\partial f}{\partial w} \quad (32)$$

where

$$net-input = f(u_1^k, u_2^k, \dots, u_p^k; w_1^k, w_2^k, \dots, w_p^k) .$$

here, the superscript k indicates the layer number and subscript p indicates the index of the input nodes.

$$\frac{\partial E}{\partial w} = \frac{\partial E}{\partial a} \cdot \frac{\partial a}{\partial f} \cdot \frac{\partial f}{\partial w} \quad (33)$$

To show the learning rule, we will derive the computation of $\frac{\partial E}{\partial y}$ for this layer.

Using (32), and (4), the adaptive rule for m_i is derived as follows:

$$\frac{\partial E}{\partial (m_{ij})} = \frac{\partial E}{\partial a_i} \cdot \frac{\partial a_i}{\partial f_i} \cdot \frac{\partial f_i}{\partial m_{ij}} = \frac{\partial E}{\partial a_i} \cdot e^{f_i} \cdot \frac{2(u_i - m_{ij})}{\sigma_{ij}^2} \quad (34)$$

where i is the number of the node in the Input Term Node layer, and j is the number of the node in the Rule Node layer.

We have

$$\frac{\partial E}{\partial a_i} = \left(\sum_k \frac{\partial E}{\partial (net-input)^k} \cdot \frac{\partial (net-input)^k}{\partial a_i} \right) \quad (35)$$

for this particular layer;

$$\frac{\partial E}{\partial (net-input)^k} = \frac{\partial E}{\partial f_k^3} = \sigma_k^3 \quad (36)$$

where σ_k^3 is the error signal for rule node at layer three, and $k = 1, 2, \dots, 25$. Also from (9), and (10)

$$\frac{\partial (net-input)^k}{\partial a_i} = \frac{\partial f_k^3}{\partial u_i^3} = 1, \text{ if } u_i^3 = \min(\text{inputs of rule node } k), \text{ otherwise } 0.$$

Hence,

$$\frac{\partial E}{\partial a_i} = \sum_k q_k \quad (37)$$

where the summation is performed over the rule nodes that a_i feeds into. Here a_i is the output of the Input Term Nodes number i and

$$q_k = \begin{cases} \sigma_k^3, & \text{if } a_i \text{ is minimum in } k\text{th rule node's input} \\ 0, & \text{otherwise} \end{cases} \quad (38)$$

Thus, the adaptive rule of m_{ij} for this layer is

$$m_{ij}(t+1) = m_{ij}(t) - \eta \cdot \frac{\partial E}{\partial a_i} \cdot e^{f_i} \cdot \frac{2(u_i - m_{ij})}{(\delta_{ij})^2}$$

In the same manner, using (32), (4) and (37), (35), the adaptive rule of δ_{ij} is

$$\delta_{ij}(t+1) = \delta_{ij}(t) - \eta \cdot \frac{\partial E}{\partial a_i} \cdot e^{f_i} \cdot \frac{2(u_i - m_{ij})^2}{(\delta_{ij})^3} \quad (39)$$

Chapter 5

5.1 Symbolic Representation and Performance Evaluation of the LFLC

In the LFLC, the intelligent control decision of the controller is determined by both the links (or weights) of the Learning Fuzzy Logic Controller and the variables of the Fuzzy Logic membership functions. The symbolic representation of the network is described in two levels: the Low Level, and the High Level.

5.2 Low Level Symbolic Representation of the LFLC

The low level symbolic representation of the network consists of the variables of the fuzzy membership function and the links in each layer of the network. We can interpret the symbolic representation of this level as the IF part of the LFLC. For example, the mean is the center of a set of data that belongs to a cluster. The cluster method that is used in this network is Self-Organization and the variances of the fuzzy logic membership function serve as the radius of the cluster. Data that is out of the range of the radius will have no effect on the node in either the Input Term Nodes layer or the Output Term Node layer. Thus, the combination of the mean and the variance of the Fuzzy Logic membership function as an activation function of a node in the neural network enables the network to perform decision making. In an extreme case, each node in the network represents Off or On in linguistic terms for the LFLC. The Fuzzy Logic membership functions following an unsupervised learning process are shown in Fig. 3 and Fig. 4. Each Gaussian membership function has a mean (center) and a variance (width). If data are presented to these Fuzzy Logic membership functions, only one membership function will have the highest activation value.

Please note that the membership function 1-2 is not shown in Fig. 3 and Fig. 5. This is due to the fact that the mean and variance values of this membership function did not contribute to the control decision making (the mean value and variance did not change their values) for this particular node. Hence, it is not shown in the Fig. 3 and Fig. 5.

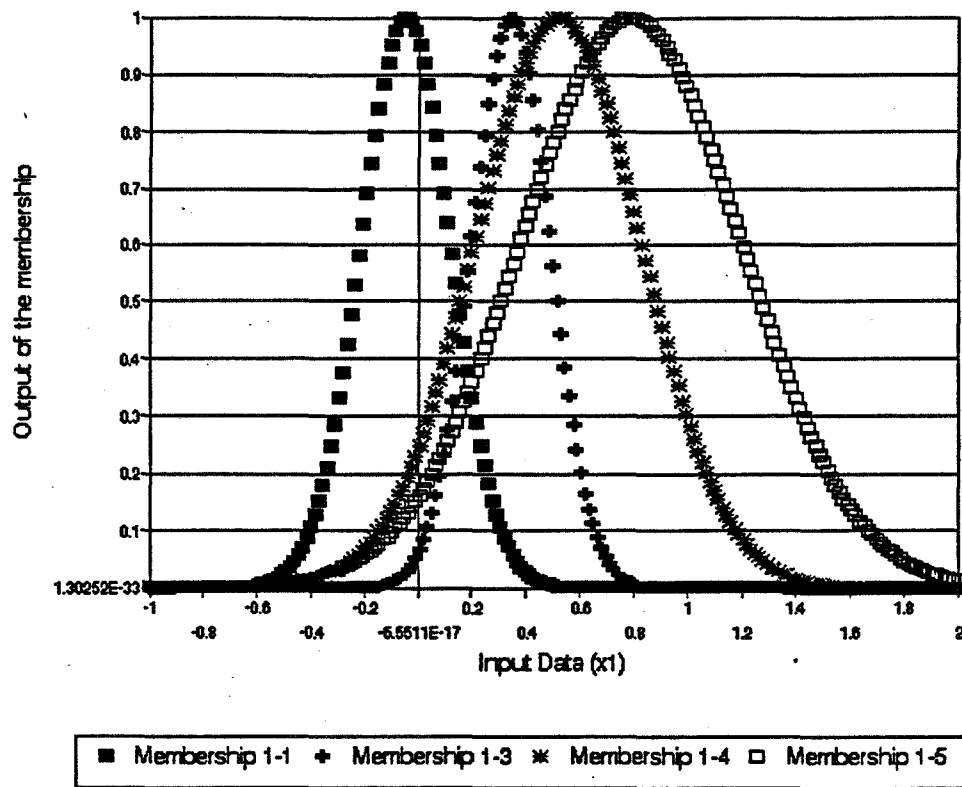


Fig.3. Initial Input Term Node membership for x_1

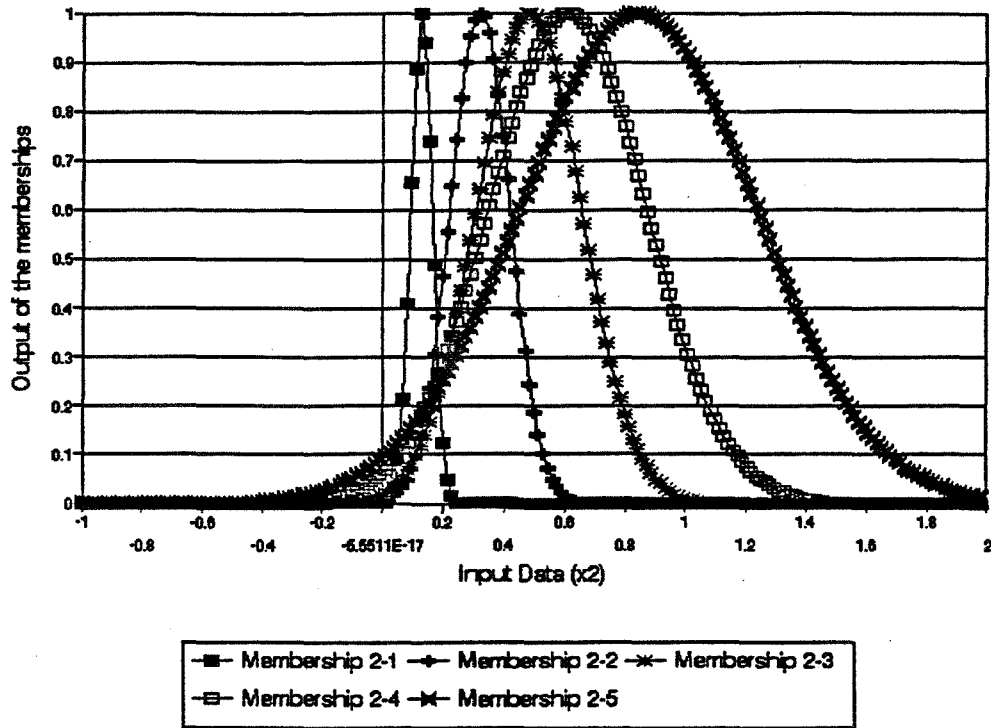


Fig.4. Initial Input Term Node membership for x_2

After supervised learning, most of the mean and variance values of the Fuzzy Logic membership functions are changed from their previous values (before the supervised learning). This is shown in Fig. 5 and Fig. 6. In Fig. 5, the mean and variance values of the Fuzzy Logic membership functions are different than those of the membership functions shown in Fig. 3. In the same manner, the mean and variance of membership functions 2-1, 2-2, 2-3, 2-4 and 2-5 in Fig. 6 are different than those of the membership functions shown in Fig. 4. This is due to the fact that in the supervised learning phase, the error between the calculated value and the desired value is calculated and propagated to all layers in the network. As a consequence, the mean

and the variance are updated according to the learning rule (30). After supervised learning, the mean and the variance represent the best control decision for a specific node.

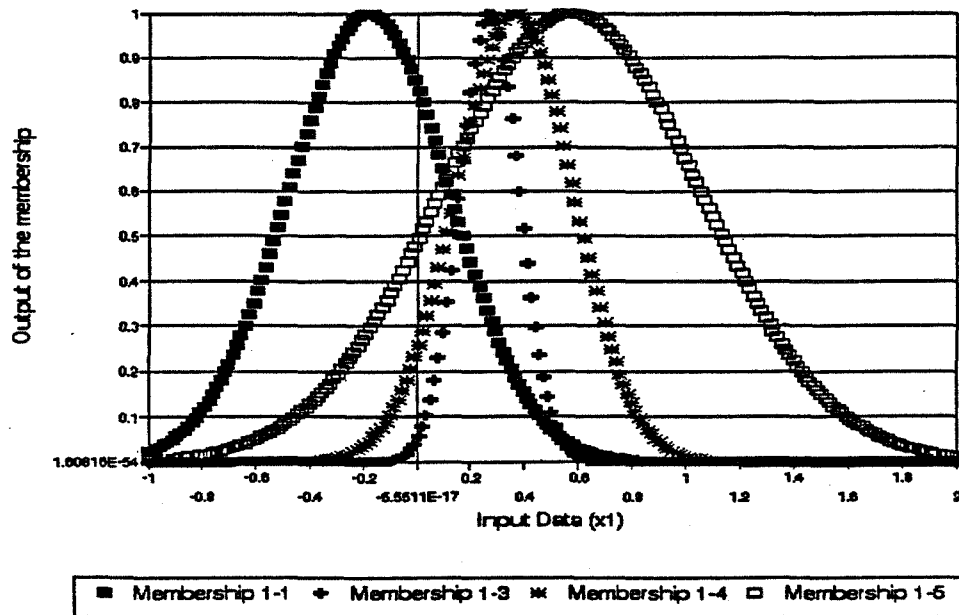


Fig.5. Final Input Term Node membership for x_1

Also, the variances of the Fuzzy Logic membership functions 1-3 and 1-4 in Fig. 3 changed their values from large value to small that is shown in Fig. 5. This can be seen better when the figure is enlarged. This indicate that only limited range of data will be in the neighbor hood of the means of these two membership functions. Otherwise, the variances of these two membership functions would increase in value, like those of the membership functions 1-1, and 1-5, to allow more data excites the membership functions that gives higher activation value. Also, the means of these two Fuzzy Logic membership functions move closer to each other. The changing of direction indicate the input data that belongs to these two clusters is somewhat separate

from other input data. Furthermore, the variances of the membership functions 1-1, and 1-5 are increased in value. This increase in value indicate that majority of the data belongs to these two clusters, and the center of these two clusters are the mean values of those two Fuzzy Logic membership functions 1-1, 1-5.

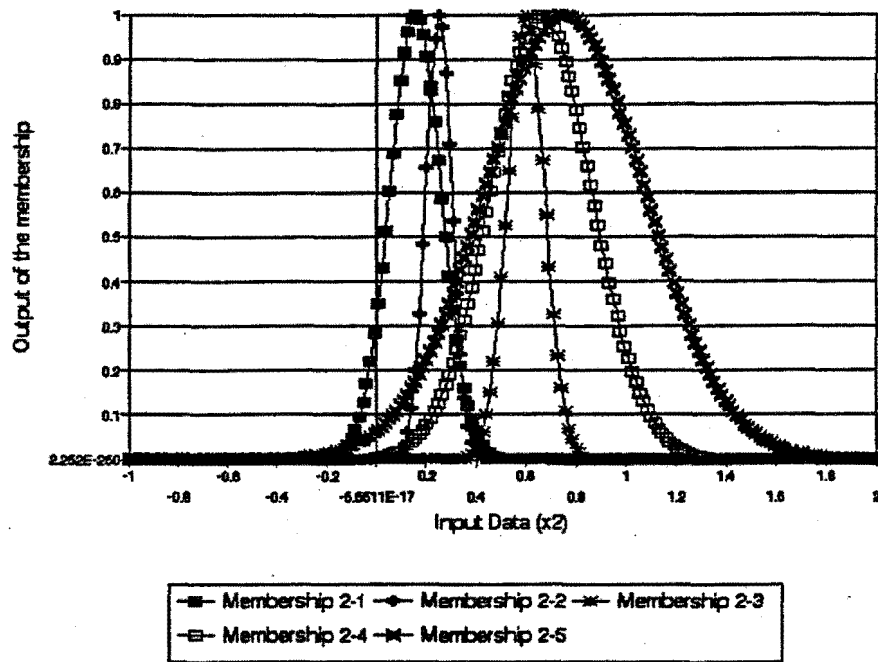


Fig.6. Final Input Term Node membership for x_2

In general, a change in the mean indicates that most of the data is close to the new value. A large value of variance indicates that a large quantity of data is located at that mean, and conversely, smaller values of the variance indicates that smaller quantities of the data are at that mean value. The convergence of the mean and the variance are shown in Fig. 7, 8, 9, and 10.

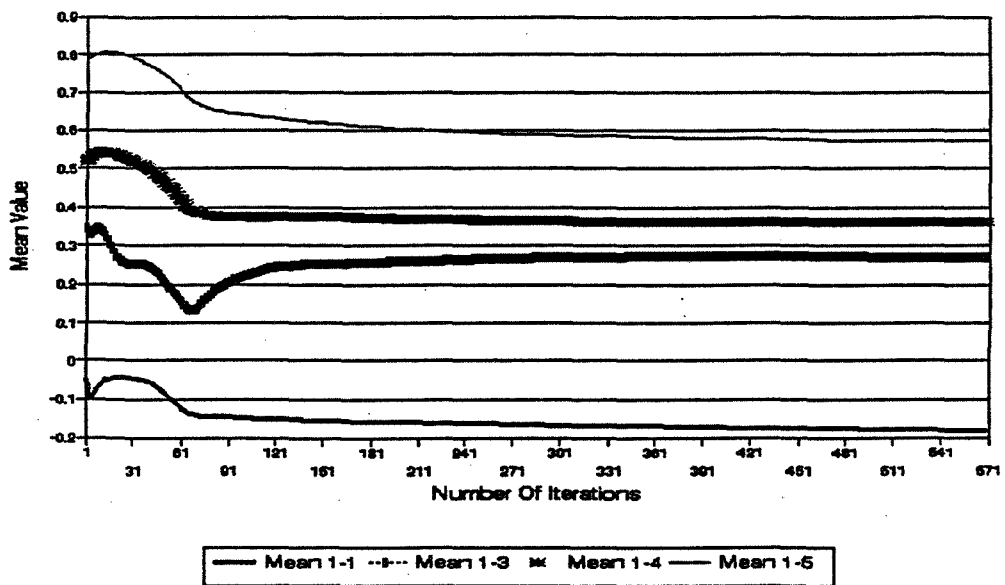


Fig.7. Convergence of the Input Term Node memberships for input x_1

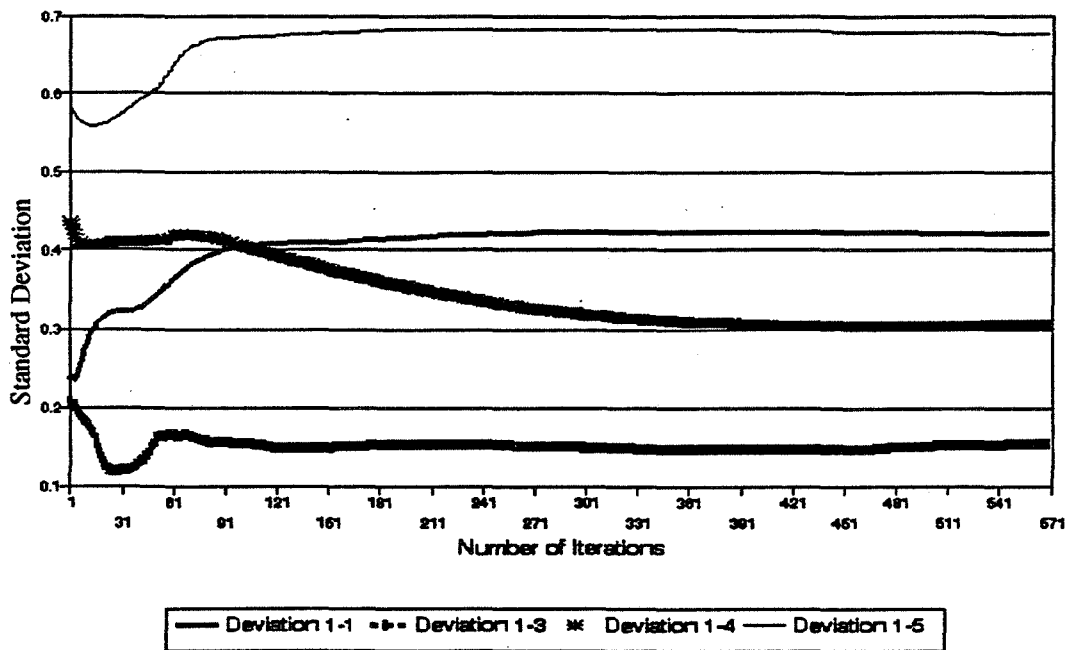


Fig.8. Convergence of the standard deviations of the Input Term Node membership for x_1

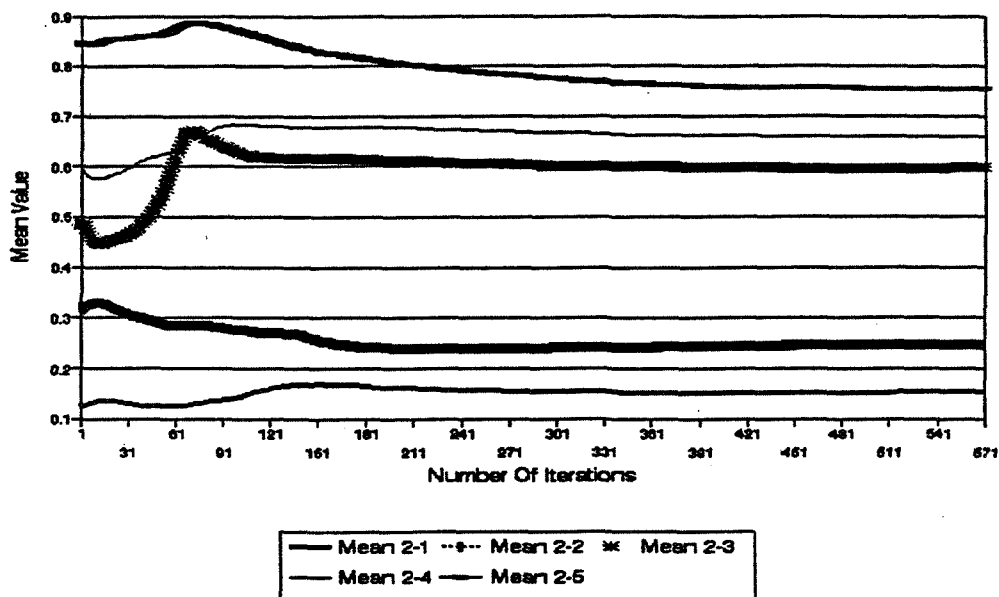


Fig.9. Convergence of the means of the Input Term Node memberships for input x_2

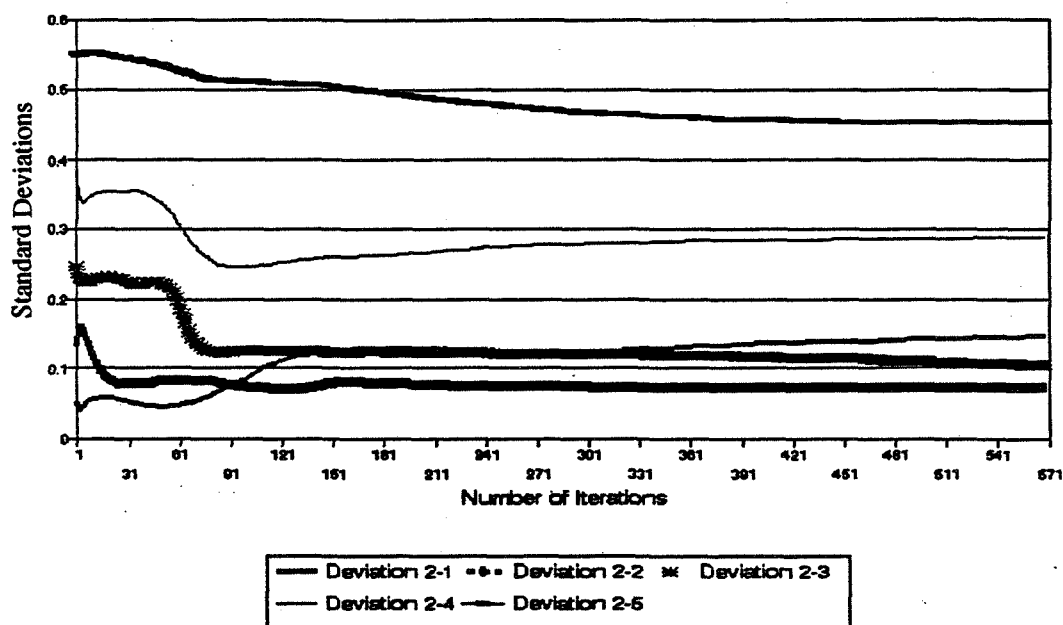


Fig.10. Convergence of the standard deviations of the Input Term Node memberships for input x_2

In a Neural Network, the links represent the firing strengths which are used to connect the nodes of one layer to the other layers. However, three types of links are used in LFLCS. The first type of link is used to transmit information from one node to another node. For example, the links of the Input Linguistic Node layer are used as a connection to transmit information from this layer to a node at the Input Term Node layer. The connection of the network is shown in Fig. 2. In the same manner, links at the Output Term Nodes are used to transmit signals from the nodes in this layer to the nodes at of Output Linguistic layer, and the links of these layers are fully connected with weights equal to 1.

In this particular network, the second type of link is constructed in such a way that for each node in the Rule Nodes layer there are only two connections from the Input Term Nodes layer. These links can then be interpreted as the IF part of the Fuzzy Logic rule. The predefined structure of these links is explained in detail in section 3.4.

The third type of link used is at the Output Term Node layer. These links are initially fully connected, but the weights are modified by the learning rule (eq. 23) in the first (unsupervised) phase of the learning process. These links can be interpreted as the THEN (consequence) part of the Fuzzy Logic rule. As can be seen in Table 1, a rule node can be connected to only one node at the Output Term Nodes layer. These links represent the firing strength for a node in that network. Furthermore, Table 1 shows that nodes number 3, 4 and 5 of the Output Term Node layer are eliminated. This is because none of the rule nodes connects to these Output Term nodes. Thus, any rule node that contains one of the three nodes from the Output Term Node, mentioned above as its consequence node, is automatically eliminated.

5.3 High Level Representation of the LFLC

In Fuzzy Logic symbolic representation of control strategy, the use of a linguistic term offers advantages over the conventional approach to specifying the control algorithm as an equation, especially, in ill-structured situations. The concept involves using a linguistic rule to describe the operation of the process from a human point of view and to capture the essential knowledge of the operation of that process, which the operator has presumably acquired through direct experience with and actual operating process. It follows that this knowledge can be used as the best rule set which the operator can obtain for the control action in linguistic terms.

In this thesis, the linguistic term of the control action is learned through the training of the network, and the teaching pattern is used for operator's knowledge. This knowledge is stored in the network by first using the Self-Organizing learning method and then using the Backpropagation method of learning in order to determine the Fuzzy Logic rule nodes.

For example, the linguistic term for the first node in the Input Term Nodes layer is interpreted as Negative Large (*NL*). The second node is interpreted as Negative Small (*NS*), the third node is interpreted as Zero (*ZN*), the fourth node is interpreted as Positive Small (*PS*), and the fifth node as Positive Large (*PL*).

The linguistic term for the first node of the Output Term Nodes is interpreted as the Negative Large (*NL*) node. The second node is interpreted as Negative Medium (*NM*), the third nodes can be interpreted as Negative Small (*NS*), the fourth node is interpreted as Zero (*ZN*), the fifth node is interpreted as Positive Small (*PS*), the sixth node

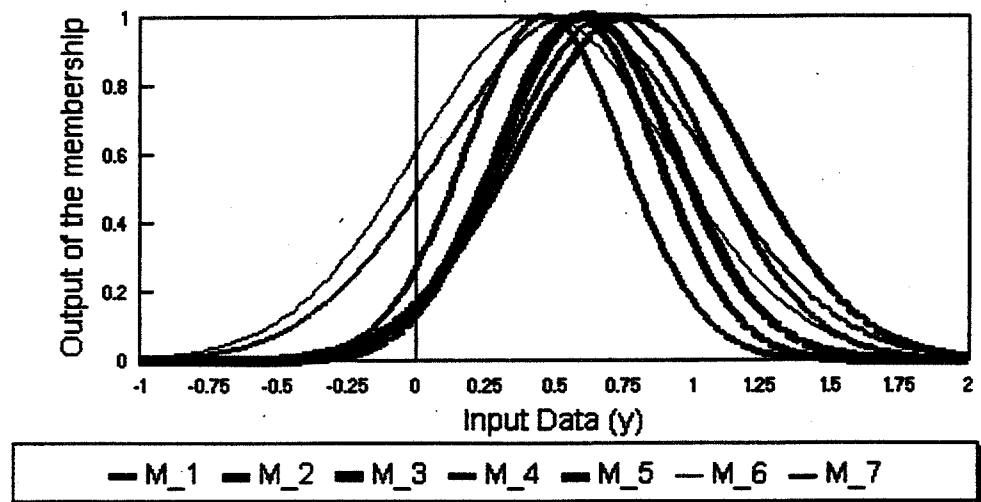


Fig.11. Initial Output Term Node membership

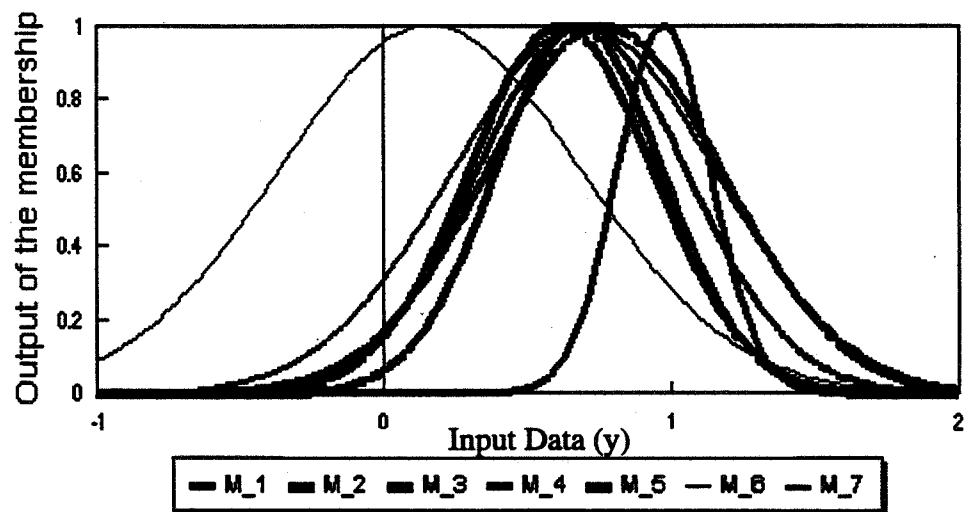


Fig.12. Final Output Term Node membership

5.4 Performance evaluation

In this section, performance of the LFLC is evaluated by an example using the LFLC in a closed loop control system. First, a conventional PI controller is designed to control the shaft position of a D.C. motor. The transfer function of the D.C. plant $G_p(z)$ is

$$G_p(z) = \frac{179.45 \times 10^{-6} (z + 1)}{(z - 1)^2} \quad (40)$$

The transfer function of a position sensor $H(z)$ which is used in the feedback path of the closed loop system is

$$H(z) = \frac{99.3z - 95.3}{z + 1} \quad (41)$$

A unit step signal is used as the input signal to the system.

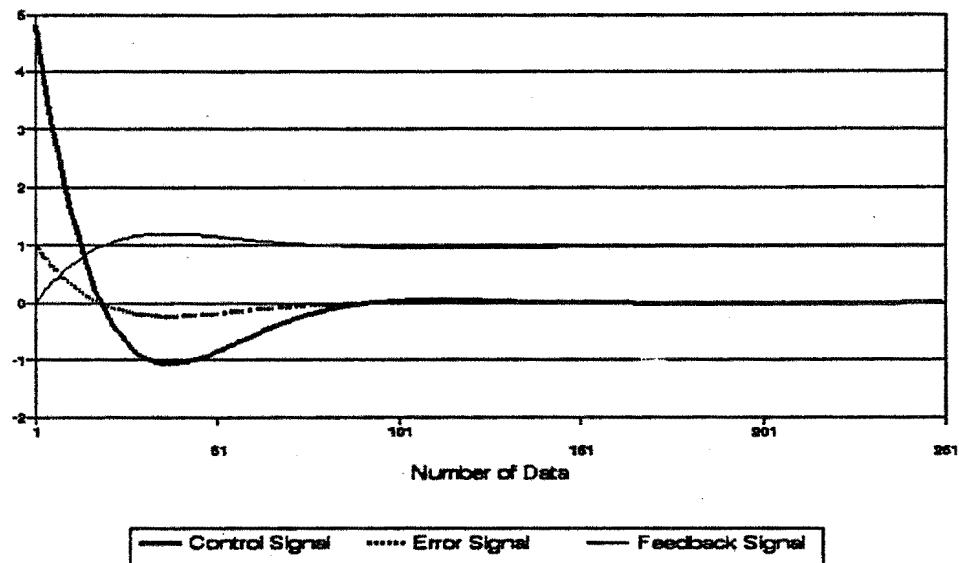


Fig.13. Training Data

The control system is implemented by simulation to obtain training data set as shown in Fig.13. This data set is then used in the off-line training of the LFLC to obtain the fuzzy logic controller, which will give approximately the same performance as a PI controller.

A closed loop control system is also simulated to illustrate the capability of the LFLC. This closed loop control system is shown in Fig. 14. In the diagram, the error signal and the feedback signal are normalized before being fed into the LFLC. This normalization process is a necessary step before any data is presented to the LFLC. This follows since during the learning process the training data is also normalized by equation (1). However, it should be observed that the control signal generated from the LFLC in Fig. 14 is going through an inverted normalization processes in order to retrieve the actual control signal. This is required since the teaching pattern is normalized by equation (2) during the training process. Equation (43) is used in the inverse normalization process.

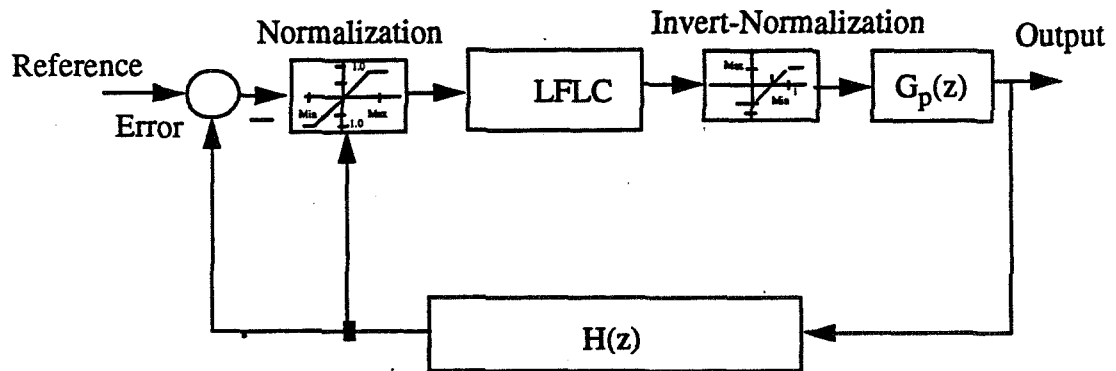


Fig.14. LFLCS with PI Fuzzy Logic Controller

(42)

$$y_{ACTUAL} = (y_{LFLC} - 0.5) \cdot (2 \times \max(|\max y_d|, |\min y_d|)) \quad (45)$$

where y_{LFLC} is the output signal from the LFLC, and y_{ACTUAL} is the actual control signal.

The output of the computer simulation of the LFLC with a PI Fuzzy Logic controller is shown in Fig. 15. The result of the simulation shows that the LFLC is able to control the D. C. motor at least as good as a conventional PI controller or even better. The LFLC shows that it has better rise time performance, better settling time and a smaller steady state error.

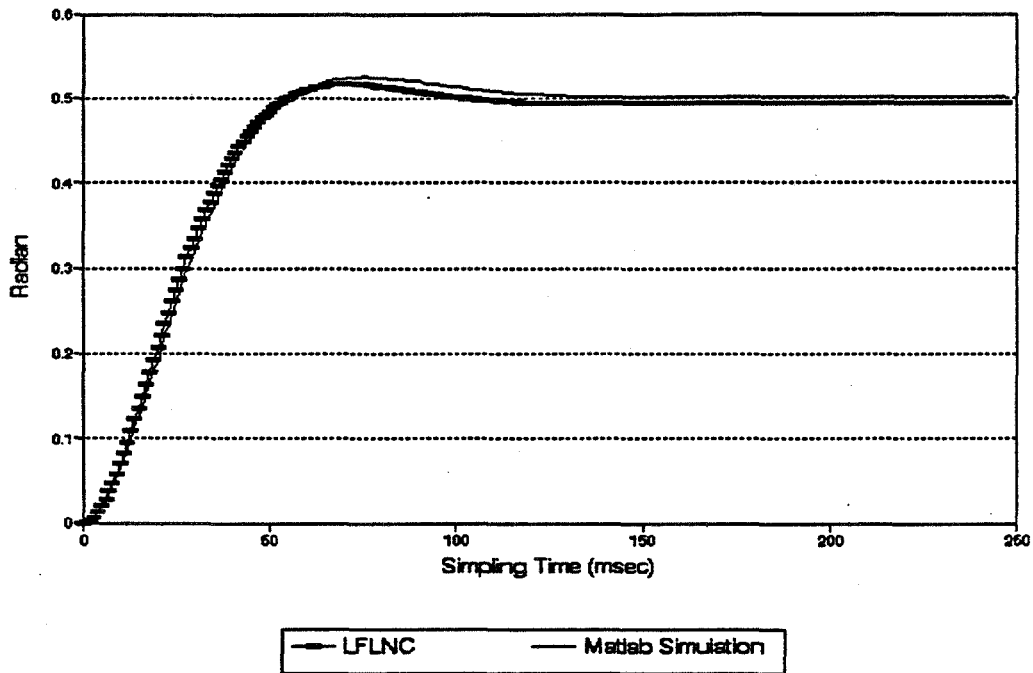


Fig.15. Step response of the system with classical and fuzzy logic PI controller

Chapter 6

6.1 Summary

In this thesis, the Learning Fuzzy Logic Controller is being developed and replaces the conventional PI controller that is used in a closed loop position control system. The result of two simulated control systems is given in Fig. 15. We find that the rise time, settling time and steady state error of the system with the LFLC are superior over those of a system with a conventional PI controller.

The Learning Fuzzy Logic Controller developed in this thesis shows that by combining the Neural Network learning concept and the Fuzzy Logic rule base, it is possible to eliminate the need for an accurate model of the system.

However, during the research of the LFLCS we discovered that there are some limitations of this approach. This is due to the need of the teaching pattern necessary to train the network. In most cases, this teaching pattern is the control signal to the plant. This signal can be either generated by computer simulation as we did in this thesis or using an actual control signal of a real system. However, in control application, this signal is not always obtainable. In order to overcome this problem in control design, Reinforcement learning approach for control design should be considered. The reinforcement learning utilizes both the knowledge of the system at hand and error prediction scheme to modify the parameters of the controller.

Appendix A

Code Functions Index to Appendix B

main()	45
open_files(void)	47
unsupervised_learning(void)	49
supervise_learning(void)	53
get_downup_act4(unsigned int input_index)	56
get_downup_act4(void)	58
backpro_forward(unsigned int index)	60
change_w4(void)	64
close_loop_control(void)	66
close_loop_backpro_forward(void)	70
close_loop_get_act2(unsigned int input_lingui_no, float inputx)	73
close_loop_activation_l2(float inputx, unsigned int input_lingui_no)	74
close_loop_l2_membership(float mean, float deviation, float u)	75
close_loop_get_act3(unsigned int input_lingui_no1, unsigned int input_lingui_no2)	76
close_loop_get_downup_act4(void)	78
find_input_deviation(unsigned int input_lingui_no)	80
find_output_deviation(unsigned int output_lingui_no)	82
error_backpro_layer2(float lrate, int input_index)	84
rule_connection(unsigned int input_term_membership, unsigned int input_lingui_no)	86
membership_connection()	88
find_next_rule_node()	90
error_backpro_layer3(void)	91
error_backpro_layer4(float *Sum_width_act, float *Sum_mean_width_act)	93
error_backpro_layer5(float lrate, unsigned int index)	95
rule_connection(unsigned int input_term_membership, unsigned int input_lingui_no)	99
membership_connection()	100
find_next_rule_node()	102
update_input_l2_w(float inputx, float alpha, unsigned int input_ling_node)	104

Code Functions Index to Appendix B

fi_l2_mini_index(float inputx, unsigned int input_ling_node)	105
update_output_w(float inputy, float alpha, unsigned int output_ling_node)	107
fi_output_mini_index(float inputy, unsigned int output_ling_node)	108
get_means(void)	110
find_mean(float max_range, float min_range, unsigned int num_membership, unsigned int input_no).....	112
initials_all(void).....	119
init_globle(void)	122
init_weight(void)	122
init_maxmin_range(void)	123
init_weight2(void)	125
init_weight3(void)	126
init_weight4(void)	126
get_act2(void)	128
activation_l2(float inputx, unsigned int input_lingui_no)	128
l2_membership(float mean, float deviation, float u)	130
get_act2(unsigned int input_lingui_no, unsigned int input_index)	131
activation_l2(float inputx, unsigned int input_lingui_no)	132
l2_membership(float mean, float deviation, float u)	133
get_act3(unsigned int input_lingui_no1, unsigned int input_lingui_no2).....	134
get_act4(unsigned int output_lingui_no, unsigned int index)	136
activation_l4(float output, unsigned int output_lingui_no)	137
l4_membership(float mean, float deviation, float u)	138
find_max_w4(void)	139
find_output(void).....	142
init_phase1(void).....	146
save_w4(void).....	150
save_w4(void).....	151
update_w(int prt_index).....	152

Code Functions Index to Appendix B

main(void).....156
read_input(void)164

APPENDIX B

Simulation Code

```
#include <stdio.h>
#include <stdlib.h>
#include "c:\borlandc\file\thesis.h"
#include "c:\borlandc\file\globalex.h"

int
main()
{
    unsigned int input_ling_no,
        output_ling_no,
        i,
        j;

    if (!open_files())
    {
        printf("CAN NOT OPEN FILE\n");
    }

    if (!read_input())
    {
        printf("CAN NOT READ INPUT FILE\n");
    }

    if (!unsupervised_learning())
    {
        printf("INITIAL PHASE1 FALSE\n");
        return (FALSE);
    }

    if (!supervise_learning())
    {
        printf("BACKPROP PHASE1 FALSE\n");
        return (FALSE);
    }

    printf("end of supervise learning\n");
}
```

```
getchar();  
if (!close_loop_control()  
{  
    printf("CLOSE CONTROL LOOP FALSE\n");  
    return (FALSE);  
}  
return (TRUE);  
}
```

```

#include <stdio.h>
#include "c:\borlandc\file\thesis.h"
#include "c:\borlandc\file\globalex.h"

int
open_files(void)
{
    if ((Error_fptr = fopen("error.txt", "r")) == NULL)
    {
        fprintf(stderr, "CAN NOT OPEN ERROR FILE\n");
        return (FALSE);
    }

    if ((Fback_fptr = fopen("fback.txt", "r")) == NULL)
    {
        fprintf(stderr, "CAN NOT OPEN FBACK FILE\n");
        return (FALSE);
    }

    if ((Consigna_fptr = fopen("consigna.txt", "r")) == NULL)
    {
        fprintf(stderr, "CAN NOT OPEN CONSIGNA FILE\n");
        return (FALSE);
    }

    if ((Output_fptr = fopen("output.txt", "w")) == NULL)
    {
        fprintf(stderr, "CAN NOT OPEN OUTPUT FILE\n");
        return (FALSE);
    }

    if ((W_l4_fptr = fopen("weigh4.txt", "w")) == NULL)
    {
        fprintf(stderr, "CAN NOT OPEN WEIGHT4 FILE\n");
        return (FALSE);
    }

    if ((Input_term_Deviation_fptr = fopen("input_D.txt", "w")) == NULL)
    {
        fprintf(stderr, "CAN NOT OPEN INPUT DEVIATION FILE\n");
        return (FALSE);
    }

    if ((Final_input_term_deviation_fptr = fopen("fi_inD.txt", "w")) == NULL)

```

```

    {
        fprintf(stderr, "CAN NOT OPEN FINAL INPUT DEVIATION FILE\n");
        return (FALSE);
    }

    if ((Output_term_Deviation_fptr = fopen("output_D.txt", "w")) == NULL)
    {
        fprintf(stderr, "CAN NOT OPEN OUTPUT DEVIATION FILE\n");
        return (FALSE);
    }

    if ((Final_output_term_deviation_fptr = fopen("fi_outD.txt", "w")) == NULL)
    {
        fprintf(stderr, "CAN NOT OPEN FINAL OUTPUT DEVIATION FILE\n");
        return (FALSE);
    }

    if ((Mean_output_fptr = fopen("mean_out.txt", "w")) == NULL)
    {
        fprintf(stderr, "CAN NOT OPEN MEAN OUTPUT FILE\n");
        return (FALSE);
    }

    if ((Final_mean_output_fptr = fopen("fin_mout.txt", "w")) == NULL)
    {
        fprintf(stderr, "CAN NOT OPEN FINAL MEAN OUTPUT FILE\n");
        return (FALSE);
    }

    if ((Mean_input_fptr = fopen("mean_in.txt", "w")) == NULL)
    {
        fprintf(stderr, "CAN NOT OPEN MEAN_IN FILE\n");
        return (FALSE);
    }

    if ((Final_mean_input_fptr = fopen("fin_min.txt", "w")) == NULL)
    {
        fprintf(stderr, "CAN NOT OPEN FINAL MEAN INPUT FILE\n");
        return (FALSE);
    }

    printf("OPEN SUCCESSFUL\n");
    return (TRUE);
}

```

```
#include <stdio.h>
#include <stdlib.h>
#include "c:\borlandc\file\thesis.h"
#include "c:\borlandc\file\globalex.h"
```

```
/******
```

```
* function init_phase1()
*
* This function excute all initial procedure for getting the center of the
* membership and the deviation of input and output term nodes. So after this
* function the input signal can be pass through the membership on layer3 and
* layer4.
*
* Input parameter: inputx1,
* inputx2.
*
* Return value: TRUE if each of the center of membership and the deviation
* are calculated.
* FALSE otherwise.
*
* Programmer:
* Leung Kam Lung 9/30/93MS. Thesis.
*
*/
```

```
int
unsupervised_learning(void)
{
    int i;
    float time,
          time2;
    unsigned int input_lingui_no,
                output_lingui_no;

    time = 0.001;
    time2 = 0.001;
    initials_all();
    input_lingui_no = 0;
    output_lingui_no = 0;
    while ((time > 0.000001) || (time2 > 0.000001))
    {
        for (i = 0; i < MAX_INPUT_INDEX; i++)
        {
            update_input_l2_w(Inputx[input_lingui_no][i], time, input_lingui_no);

```



```

        update_input_l2_w(Inputx[input_lingui_no + 1][i], time, input_lingui_no + 1);
        update_output_w(Output[output_lingui_no][i], time2, output_lingui_no);
    }
    time = time * 0.995;
    time2 = time2 * 0.995;

}

if (!save_mean_input())
{
    printf("Mean_input FALSE\n");
}

if (!save_mean_output())
{
    printf("Mean_output FALSE\n");
}

/*
 * up to now only the center(mean) of the input and output membership are
 * calculated
 */

for (input_lingui_no = 0; input_lingui_no < INPUT_LINGUI_NO; input_lingui_no++)
{
    if (find_input_deviation(input_lingui_no))
    {
        printf(" Deviation of each Membership is find for INPUT_TERM_NODE %d\n",
input_lingui_no);
    }
    else
    {
        printf(" NOT Deviation of each Membership is find for INPUT_TERM_NODE %d\n",
input_lingui_no);
        return (FALSE);
    }

}

/* End for input_lingio_no */

if (!save_input_deviation())
{
    printf("CAN NOT SAVE INPUT DEVIATION\n");
}

```

```

for (output_lingui_no = 0; output_lingui_no < OUTPUT_LINGUI_NO; output_lingui_no++)
{
    if (find_output_deviation(output_lingui_no))
    {
        printf(" Deviation of each Membership is find for OUTPUT_TERM_NODE %d\n",
output_lingui_no);
    }
    else
    {
        printf(" NOT Deviation of each Membership is find for OUTPUT_TERM_NODE %d\n",
output_lingui_no);
        return (FALSE);
    }
}

/* End for output_lingio_no */

if (!save_output_deviation())
{
    printf("CAN NOT SAVE OUTPUT DEVIATION\n");
}

/*
* now all deviation of each membership for input term nodes and output
* term nodes are calculated
*/

if (!update_w_l4())
{
    printf("update_w_l4 FALSE\n");
    return (FALSE);
}

/* printf(" after updata w 4\n"); */

if (!find_max_w4())
{
    printf("can not find max weight 4 FALSE\n");
    return (FALSE);
}

/* printf(" after find_max_w4\n"); */

```

```
if (!save_w4())
{
    printf("can not save weight 4\n");
    return (FALSE);
}

/* printf(" after save w 4\n"); */

return (TRUE);
}
```

```

#include <stdio.h>
#include <stdlib.h>
#include "c:\borlandc\file\thesis.h"
#include "c:\borlandc\file\globalex.h"

int
supervise_learning(void)

{
    float lrate;
    int index,
        retv5,
        retv4;

    lrate = 0.045;

    /*
     * forward phase of the learning for all inputs. At this point all
     * Errors will be calculated and then updata procedures will then
     * execute. Error will be sumed for smoth result.
     */

    Cal_error = 1.0;

    while ((lrate > MIN_LRATE) && (Cal_error > 0.025))
    {

        Cal_error = MIN_ERROR;
        Max_indexs = 0;

        for (index = 0; index < MAX_INPUT_INDEX; index++)
        {

            /* printf("NOW IN THE BACKPRO FORWARD \n"); */
            if (!backpro_forward(index))
            {
                printf(" Can not find the output from backpro forward\n");
                return (FALSE);
            }

            /* printf("NOW IN THE BACKPRO\n"); */
            /* getchar(); */

            if (!error_backpro_layer5(lrate, index))
            {
                printf(" Can not find the error_backpro_layer5_ptr\n");
            }
        }
    }
}

```

```

        return (FALSE);
    }

    /* printf("\n after LAYER 5\n\n"); */

    if (!error_backpro_layer4(Sum_width_act, Sum_mean_width_act))
    {
        printf(" Can not find the error_backpro_layer4_ptr\n");
        return (FALSE);
    }

    /* printf("\n after LAYER 4\n\n"); */

    if (!error_backpro_layer3())
    {
        printf(" Can not find the error_backpro_layer3_ptr\n");
        return (FALSE);
    }

    /* printf("\n after LAYER 3\n\n"); */

    if (!error_backpro_layer2(lrate, index))
    {
        printf(" Can not find the error_backpro_layer2_ptr\n");
        return (FALSE);
    }

    /* printf("\n after LAYER 2\n\n"); */
    /* printf("\nNOW UPDATE WEIGHT\n\n"); */

    if (!update_w(index))
    {
        printf("Error update the weight\n");
        return (FALSE);
    }
    /* getchar(); */

}          /* End of for input_index */

lrate *= 0.998;

printf(" actual output %f ", Output[0][Max_indexes]);
printf(" calculated output %f\n",
        Output_down_up[0][Max_indexes]);

printf("learn rate %1.10f ", lrate);

```

```
        printf("Cal_error %1.10f\n", Cal_error);

    }          /* End of the while lrate loop */

    save_final_mean_input();
    save_final_mean_output();
    save_final_input_deviation();
    save_final_output_deviation();

    return (TRUE);
}
```

```

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#include "c:\borlandc\file\thesis.h"
#include "c:\borlandc\file\globalex.h"

int
get_downup_act4(unsigned int input_index)
{
    unsigned int output_lingui_no;
    unsigned int output_term_membership;
    unsigned int rule_index;
    float act4;

    for (output_lingui_no = 0; output_lingui_no < OUTPUT_LINGUI_NO; output_lingui_no++)
    {
        for (output_term_membership = 0; output_term_membership <
Output_membership_array[output_lingui_no];
output_term_membership++)
        {
            act4 = 0.0;

            for (rule_index = 0; rule_index < RULE_NODE; rule_index++)
            {
                act4 += Act_l3[rule_index] *
W_l4[output_lingui_no][output_term_membership][rule_index];

            }          /* End rule_index */

            if (act4 >= 1)
            {
                Act_downup_l4[output_lingui_no][output_term_membership] = 1.0;
            }

            else
            {
                Act_downup_l4[output_lingui_no][output_term_membership] = act4;
            }
        }
    }
}

```

```

    }
    if (act4 > MAXFLOAT)
    {
        printf("activation value of level 4 is bigger than max float\n");
        return (FALSE);
    }

}          /* End for output_term_membership */

}          /* End for output_lingui_no */

return (TRUE);
}

```



```

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#include "c:\borlandc\file\thesis.h"
#include "c:\borlandc\file\globalex.h"

int
get_downup_act4(void)
{
    unsigned int output_lingui_no;
    unsigned int output_term_membership;
    unsigned int rule_index;
    float act4;

    for (output_lingui_no = 0; output_lingui_no < OUTPUT_LINGUI_NO; output_lingui_no++)
    {
        for (output_term_membership = 0; output_term_membership <
Output_membership_array[output_lingui_no];
        output_term_membership++)
        {
            act4 = 0.0;

            for (rule_index = 0; rule_index < RULE_NODE; rule_index++)
            {
                act4 += Act_l3[rule_index] *
W_l4[output_lingui_no][output_term_membership][rule_index];

            }          /* End rule_index */

            if (act4 >= 1)
            {
                Act_downup_l4[output_lingui_no][output_term_membership] = 1.0;
            }

            else
            {
                Act_downup_l4[output_lingui_no][output_term_membership] = act4;
            }
        }
    }
}

```

```

    }

    if (act4 > MAXFLOAT)
    {
        printf("activation value of level 4 is bigger than max float\n");
        return (FALSE);
    }

} /* End for output_term_membership */

} /* End for output_lingui_no */

return (TRUE);
}

```

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
```

```
#include "c:\borlandc\file\thesis.h"
#include "c:\borlandc\file\globalex.h"
```

```

/*****
 * function backpro()
 *
 * This function calculates the network output in downup pass of the network.
 * It call the following functions.
 *
 * get_downup_act4;
 *get_act3;
 *get_act2;
 *
 *
 * Input parameter: index
 *
 * Return value: int type.
 *
 *programmer:Leung Kam LungMS. Thesis9/28/93
 *
 */

```

```
int
backpro_forward(unsigned int index)
```

```
{
    int retval2,
        retval3,
        retval4;

    unsigned int input_lingui_no,
        input_lingui_no1,
        input_lingui_no2,
        output_lingui_no;

    float out1,
        f,
        f2,
        temp_error,
        abs_cal_error;
```

```
unsigned int output_term_membership,  
membership;
```

```
input_lingui_no1 = 0;  
input_lingui_no2 = 1;  
output_lingui_no = 0;
```

```
/* printf(" index at backpro %d\n",index); */
```

```
for (input_lingui_no = 0; input_lingui_no < INPUT_LINGUI_NO; input_lingui_no++)  
{
```

```
    retval2 = get_act2(input_lingui_no, index);
```

```
    if (retval2 == FALSE)
```

```
    {
```

```
        printf("reture value of act_function_ptr2 is > or < max,min double\n");
```

```
        return (FALSE);
```

```
    } /* End of if retval2 */
```

```
} /* End of for input_lingui_no */
```

```
retval3 = get_act3(input_lingui_no1, input_lingui_no2);
```

```
if (retval3 == FALSE)
```

```
{
```

```
    printf("reture value of act_function_ptr3 is > or < max,min double\n");
```

```
    return (FALSE);
```

```
} /* End of if retval3 */
```

```
retval4 = get_downup_act4();
```

```
if (retval4 == FALSE)
```

```
{
```

```
    printf("reture value of act_function_downup_ptr4 is > or < max,min double\n");
```

```
    return (FALSE);
```

```
} /* End of if retval */
```

```

/*
 * The next for loop calculates the network's output of the supervise
 * learning
 */

for (output_lingui_no = 0; output_lingui_no < OUTPUT_LINGUI_NO; output_lingui_no++)
{
    Output_down_up[output_lingui_no][index] = 0.0;
    f = f2 = 0.0;

    for (output_term_membership = 0; output_term_membership <
        Output_membership_array[output_lingui_no];
        output_term_membership++)
    {
        f += (Mean_output[output_lingui_no][output_term_membership] *
            Output_term_Deviation[output_lingui_no][output_term_membership] *
            Act_downup_l4[output_lingui_no][output_term_membership]);

        f2 += (Output_term_Deviation[output_lingui_no][output_term_membership] *
            Act_downup_l4[output_lingui_no][output_term_membership]);
    }
    /* End of output_term_membership */

    out1 = Output_down_up[output_lingui_no][index] = f / f2;

    Error_array[output_lingui_no] = temp_error = (float) (Output[output_lingui_no][index] -
        Output_down_up[output_lingui_no][index]);

    /* printf(" actual output %f ", Output[0][index]); */

    /*
    * printf(" calculated output %f, Error %fn", out1,
    * Error_array[output_lingui_no]);
    */
    /* getchar(); */

    /* abs_cal_error = (float) pow(fabs(temp_error),2); */
    abs_cal_error = (float) fabs(temp_error / Output[output_lingui_no][index]);
    if (abs_cal_error > Cal_error)
    {
        Cal_error = abs_cal_error;
    }
}

```

```
        Max_indexs = index;
    }

}          /* End of output_lingui_no */

return (TRUE);
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
```

```
#include "c:\borlandc\file\thesis.h"
#include "c:\borlandc\file\globalex.h"
```

```
int
change_w4(void)
```

```
{
    float weight4;
    unsigned int output_lingui_no;
    unsigned int output_term_membership;
    unsigned int rule_index;

    for (output_lingui_no = 0; output_lingui_no < OUTPUT_LINGUI_NO; output_lingui_no++)
    {
        for (output_term_membership = 0; output_term_membership <
Output_membership_array[output_lingui_no];
            output_term_membership++)
        {
            for (rule_index = 0; rule_index < RULE_NODE; rule_index++)
            {
                /*
                * printf("output_term_membership %d, rule
                * %d",output_term_membership, rule_index);
                */

                /*
                * printf(" weight4 %f
                * ",W_14[output_lingui_no][output_term_membership][rule_index
                * ]);
                */

                weight4 = W_14[output_lingui_no][output_term_membership][rule_index] -
Act_14[output_lingui_no][output_term_membership] *
                (Act_13[rule_index] -
                W_14[output_lingui_no][output_term_membership][rule_index]);

                /* printf("weight4 %f\n",weight4); */
            }
        }
    }
}
```

```

        if (weight4 > MAXFLOAT)
        {
            printf("weight4 value of output_lingui_no %d output term membership %d rule_index &d
is > or < expected value\n",
                output_lingui_no, output_term_membership, rule_index);

            return (FALSE);
        }
    }
    /* End rule_index */
}
/* End for output_term_membership */
}
/* End for output_lingui_no */
return (TRUE);
}
/* End of function change_w4 */

```



```

#include <stdio.h>
#include <stdlib.h>
#include "c:\borlandc\file\thesis.h"
#include "c:\borlandc\file\globalex.h"

```

```

int
close_loop_control(void)

```

```

{

```

```

    int index,
        time;
    float inputx1,
        inputx2,
        Fback;
    float y,
        y1,
        y2;
    float u1,
        u2;
    float fb,
        fb1;
    float close_input1,
        close_input2;
    float inv_control_signal;
    float plant_output,
        reference;

```

```

    y1 = 0.0;
    y2 = 0.0;
    u1 = 0.0;
    u2 = 0.0;
    fb = 0.0;
    fb1 = 0.0;

```

```

    reference = 1.0;

```

```

    plant_output = 0.0;

```

```

    close_input1 = reference - fb;
    close_input2 = fb;

```

```

    Close_loop_input[0] = (close_input1 * Input_slope1);
    Close_loop_input[1] = (close_input2 * Input_slope2);

```

```

printf("IN CONTROL LOOP\n");

for (time = 2; time < MAX_TIME; time++)
{
    printf(" Error %8f\n", close_input1);
    printf(" Fback %8f\n", close_input2);
    printf(" Input_x0 %8f\n", Close_loop_input[0]);
    printf(" Input_x1 %8f\n", Close_loop_input[1]);

    if (!close_loop_backpro_forward())
    {
        printf(" Can not find the output from close loop backpro forward\n");
        return (FALSE);
    }

    inv_control_signal = ((Control_signal - 0.5) / Output_slope1);

    printf("Teach %1.6f\n", inv_control_signal);

    plant_output = 2 * y1 - y2 + 179.45e-6 * (u1 + u2);

    printf("output of the plant %f\n", plant_output);

    fb = 99.3 * plant_output - 95.3 * y1 - fb1;
    printf(" fback %f\n", fb);

    close_input1 = reference - fb;
    close_input2 = fb;

    /*
    * printf("Close input1 bigger %f\n",close_input1*Input_slope1);
    * printf("Close input2 bigger %f\n",close_input2*Input_slope2);
    */

    if ((close_input1 * Input_slope1) > Max_input1)
    {
        Close_loop_input[0] = Max_input1;
    }
    else
    {

```

```

if ((close_input1 * Input_slope1) < Min_input1)
{
    Close_loop_input[0] = Min_input1;
}
else
{
    Close_loop_input[0] = (close_input1 * Input_slope1);
}
}

if ((close_input2 * Input_slope2) > Max_input2)
{
    printf("Max_input2 %f\n", Max_input2);
    Close_loop_input[1] = Max_input2;
}
else
{
    if ((close_input2 * Input_slope2) < Min_input2)
    {
        printf("Min_input2 %f\n", Min_input2);
        Close_loop_input[1] = Min_input2;
    }
    else
    {
        Close_loop_input[1] = (close_input2 * Input_slope2);
    }
}

u2 = u1;
u1 = inv_control_signal;
y2 = y1;
y1 = plant_output;
fb1 = fb;

/*
* printf(" u2 %8f\n",u2); printf(" u1 %8f\n",u1); printf(" y2 %8f
* \n",y2); printf(" y1 %8f\n",y1); printf(" fb1 %8f\n",fb1);
*/

getchar();

```

```
}  
    fclose(Output_fptr);  
    fclose(Max_data_fptr);  
  
    return (TRUE);  
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
```

```
#include "c:\borlandc\file\thesis.h"
#include "c:\borlandc\file\globalex.h"
```

```
/******
```

```
 * function close_loop_backpro_forward ()
```

```
 *
```

```
 * This function calculates the network output in downup pass of the network.
```

```
 * It call the following functions.
```

```
 *
```

```
 * close_get_downup_act4;
```

```
 *close_get_act3;
```

```
 *close_get_act2;
```

```
 *
```

```
 *
```

```
 * Input parameter: index
```

```
 *
```

```
 * Return value: int type.
```

```
 *
```

```
 *programmer:Leung Kam LungMS. Thesis9/28/93
```

```
 *
```

```
 */
```

```
int
```

```
close_loop_backpro_forward(void)
```

```
{
```

```
    int retval2,
```

```
        retval3,
```

```
        retval4;
```

```
    unsigned int input_lingui_no,
```

```
        input_lingui_no1,
```

```
        input_lingui_no2,
```

```
        output_lingui_no;
```

```
    float out1,
```

```
        f,
```

```
        f2,
```

```
        temp_error,
```

```
        abs_cal_error;
```

```
unsigned int output_term_membership,  
    membership;
```

```
input_lingui_no1 = 0;  
input_lingui_no2 = 1;  
output_lingui_no = 0;
```

```
for (input_lingui_no = 0; input_lingui_no < INPUT_LINGUI_NO; input_lingui_no++)  
{
```

```
    retval2 = close_loop_get_act2(input_lingui_no, Close_loop_input[input_lingui_no]);
```

```
    if (retval2 == FALSE)
```

```
    {  
        printf("return value of close_loop_get_act2 is > or < max,min double\n");  
        return (FALSE);  
    } /* End of if retval2 */
```

```
} /* End of for input_lingui_no */
```

```
retval3 = close_loop_get_act3(input_lingui_no1, input_lingui_no2);
```

```
if (retval3 == FALSE)
```

```
{  
    printf("return value of close_loop_get_act3 is > or < max,min double\n");  
    return (FALSE);  
} /* End of if retval3 */
```

```
retval4 = close_loop_get_downup_act4();
```

```
if (retval4 == FALSE)
```

```
{  
    printf("return value of close_loop_get_downup_act4 is > or < max,min double\n");  
    return (FALSE);  
} /* End of if retval */
```

```
/*
```

```
 * The next for loop calculates the network's output of the supervise
```

```
 * learning
```

```
*/
```

```

for (output_lingui_no = 0; output_lingui_no < OUTPUT_LINGUI_NO; output_lingui_no++)
{
    f = f2 = 0.0;

    for (output_term_membership = 0; output_term_membership <
Output_membership_array[output_lingui_no];
        output_term_membership++)
    {
        f += (Mean_output[output_lingui_no][output_term_membership] *
Output_term_Deviation[output_lingui_no][output_term_membership] *
        Act_downup_14[output_lingui_no][output_term_membership]);

        f2 += (Output_term_Deviation[output_lingui_no][output_term_membership] *
        Act_downup_14[output_lingui_no][output_term_membership]);
    }
    /* End of output_term_membership */

    Control_signal = f / f2;

    if ((Control_signal > MAXFLOAT) || (Control_signal < MINFLOAT))
    {
        printf("Control_signal OUT OF FLOATING POINT RANGE\n");
        return (FALSE);
    }
    else
    {
        printf("Output1 %1.6f\n", Control_signal);
    }
}
/* End of output_lingui_no */

return (TRUE);
}

```

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
```

```
#include "c:\borlandc\file\thesis.h"
#include "c:\borlandc\file\globalex.h"
```

```
/******
```

```
* function close_loop_get_act2()
*
* This function calculates the output of the layer two of the network.
*
* Input parameter: input_lingui_no, inputx.
*
* Return value: int type.
*
* programmer: Leung Kam Lung MS. Thesis 9/28/93
*
*/
```

```
int
close_loop_get_act2(unsigned int input_lingui_no, float inputx)
{
    int retval;

    retval = close_loop_activation_l2(inputx, input_lingui_no);
    if (retval == FALSE)
    {
        printf("return value of act_function_ptr is > or < max,min double\n");
        return (FALSE);
    }
    /* End of if retval */

    return (TRUE);
}
```

```
/
*****
```

```
*****
* This function is called by get_act2 to set up the parameters for calculates the output value
* for layer of the network.
*
* Input parameter: inputx,
```



```

*input_lingui_no.
*
* Return value: TRUE if successfull calculate the output values.
* FALSE otherwise.
*
*
*programmer:Leung Kam LungMS. Thesis9/28/93
*/

int
close_loop_activation_l2(float inputx, unsigned int input_lingui_no)

{
    int membership;
    float temp;

    for (membership = 0; membership < Input_membership_array[input_lingui_no]; membership++)
    {
        temp = Act_l2[input_lingui_no][membership] - (float)
close_loop_l2_membership(Mean_input[input_lingui_no][membership],
                        Input_term_Deviation[input_lingui_no][membership],
                        inputx);

        printf("a_2 %f ", temp);

        if (temp >= MAXFLOAT)
        {
            printf("close_loop Act_l2 value of input_lingui_no %d membership %d is > or < expected
value\n",
                input_lingui_no, membership);

            return (FALSE);
        }
    }

    /* End of for membership */
    printf("\n");

    getchar();
}

```

```

    return (TRUE);
}          /* End of activation_12 */

/*****
* The function calculates the activation value for membership function in
* layer 2 of the NNFLC.
* input parameters:
* mean
* deviation
* u(input value)
* output:
* Return type double
*
* programmer: Leung Kam Lung MS. Thesis 9/28/93
*/

double
close_loop_12_membership(float mean, float deviation, float u)
{
    /*
    * printf("2 deviation %f mean %f float %f", deviation, mean, u);
    * getchar();
    */

    return (exp(-pow(((double) u - (double) mean), 2) / (pow((double) deviation, 2))));
}

```

```

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>

#include "c:\borlandc\file\thesis.h"
#include "c:\borlandc\file\globalex.h"

/
*****
*****
* function close_loop_get_act3()
*
* This function find the activation value of the layer three by perform the minimum of
* the output of the input term nodes which are the output of the membership in layer 2.
* The activation value of this layer(rule_nodes) are the minimum value from the output
* of the layer 2.
*
* Input parameter: input_lingui_no1,
*                  input_lingui_no2.
*
*
* Return value: TRUE if successfull calculate the output values.
* FALSE otherwise.
*
*
*programmer:Leung Kam LungMS. Thesis9/28/93
*
*/

int
close_loop_get_act3(unsigned int input_lingui_no1, unsigned int input_lingui_no2)
{
    unsigned int rule_node;

    unsigned int lingno1_membership;
    unsigned int lingno2_membership;

    float smallest;
    float smallest2;

    rule_node = 0;
    for (lingno1_membership = 0; lingno1_membership < MEMBERSHIP_NO1;
        lingno1_membership++)
    {

```

```

smallest = Act_12[input_lingui_no1][lingno1_membership];

for (lingno2_membership = 0; lingno2_membership < MEMBERSHIP_NO2;
lingno2_membership++)
{
    /* printf("rule_node = %d\n", rule_node); */
    smallest2 = Act_12[input_lingui_no2][lingno2_membership];
    /* printf("smallest = %f\n", smallest); */
    /* printf("smallest2 = %f\n", smallest2); */

    if (smallest < smallest2)
    {
        Act_13[rule_node] = smallest;
        /* printf("Act_13 = %f\n", Act_13[rule_node]); */
    }
    else
    {
        Act_13[rule_node] = smallest2;
        /* printf("Act_13 = %f\n", Act_13[rule_node]); */
    }

    rule_node++;
    if (rule_node > RULE_NODE)
    {
        printf("ERROR number of rule nodes are greater than the expected value\n");
        return (FALSE);
    }
}

/* End of lingno2_membership */

/* End of lingno1_membership */

return (TRUE);
}

/* End of the program */

```

```

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#include "c:\borlandc\file\thesis.h"
#include "c:\borlandc\file\globalex.h"

int
close_loop_get_downup_act4(void)
{
    unsigned int output_lingui_no;
    unsigned int output_term_membership;
    unsigned int rule_index;
    float act4;

    for (output_lingui_no = 0; output_lingui_no < OUTPUT_LINGUI_NO; output_lingui_no++)
    {
        for (output_term_membership = 0; output_term_membership <
Output_membership_array[output_lingui_no];
            output_term_membership++)
        {
            act4 = 0.0;

            for (rule_index = 0; rule_index < RULE_NODE; rule_index++)
            {
                act4 += Act_l3[rule_index] *
W_l4[output_lingui_no][output_term_membership][rule_index];

            }          /* End rule_index */

            /*
            * printf(" act 4 %8f\n",act4); getchar();
            */

            if (act4 >= 1)
            {
                Act_downup_l4[output_lingui_no][output_term_membership] = 1.0;
            }

            else

```

```

    {
        Act_downup_l4[output_lingui_no][output_term_membership] = act4;
    }

    if (act4 > MAXFLOAT)
    {
        printf("activation value of level 4 is bigger than max float\n");
        return (FALSE);
    }
}

/* End for output_term_membership */

}

/* End for output_lingui_no */

return (TRUE);
}

```

```

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#include "c:\borlandc\file\thesis.h"
#include "c:\borlandc\file\globalex.h"

/
*****
*****
* This function find the deviation of each of the membership on layer2. The INPUT_OVERLAP
* parameter is 2.
*
*Input parameter: input_lingui_no
*
*Return value: TRUE if all deviation values are calculated
*FALSE otherwise.
*
* Programmer:Leung Kam Lung9/30/93MS. Thesis.
*/

int
find_input_deviation(unsigned int input_lingui_no)
{
    float closest_mean;
    float current_mean;
    float temp,
        temp_smallest;
    int membership1;
    int membership;

    /* printf("input ling no %d\n",input_lingui_no); */

    for (membership = 0; membership < Input_membership_array[input_lingui_no]; membership++)
    {

        current_mean = Mean_input[input_lingui_no][membership];
        /* printf(" current_mean %f\n",current_mean); */

        closest_mean = MAXFLOAT;
        for (membership1 = 0; membership1 < Input_membership_array[input_lingui_no];
membership1++)
        {

            if (membership != membership1)

```

```

    {
        temp = Mean_input[input_lingui_no][membership1];

        temp_smallest = fabs(temp - current_mean);
        /* printf("temp smallest %f\n",temp_smallest); */

        if (temp_smallest < closest_mean)
        {
            closest_mean = temp_smallest;
        }
    }

} /* End of the for membership1 */

if ((closest_mean < MINFLOAT) || (closest_mean > MAXFLOAT))
{
    return (FALSE);
}

/* printf("closest mean %f\n",closest_mean); */
Input_term_Deviation[input_lingui_no][membership] = (float) (fabs((double) (current_mean -
closest_mean)) / INPUT_OVERLAP);

/*
 * printf("Input term D
 * %8f\n",Input_term_Deviation[input_lingui_no][membership]);
 */
} /* End of the for membership */
return (TRUE);

} /* End of the function find_deviation */

/
*****
*****
* This function find the deviation of each of the membership on layer2. The OUTPUT_OVERLAP
* parameter is 2.
*
*Input parameter: input_lingui_no
*
*Return value: TRUE if all deviation values are calculated

```



```

*FALSE otherwise.
*
* Programmer:Leung Kam Lung9/30/93MS. Thesis.
*/

int
find_output_deviation(unsigned int output_lingui_no)
{
    float closest_mean;
    float current_mean;
    float temp,
        temp_smallest;
    int membership1;
    int membership;

    for (membership = 0; membership < Output_membership_array[output_lingui_no]; membership++)
    {

        current_mean = Mean_output[output_lingui_no][membership];
        /* printf(" current_mean %fn",current_mean); */

        closest_mean = MAXFLOAT;
        for (membership1 = 0; membership1 < Output_membership_array[output_lingui_no];
membership1++)
        {

            if (membership != membership1)
            {

                temp = Mean_output[output_lingui_no][membership1];

                temp_smallest = fabs(temp - current_mean);
                /* printf("temp smallest %fn",temp_smallest); */

                if (temp_smallest < closest_mean)
                {

                    closest_mean = temp_smallest;

                }
            }
        }
    }

    /* End of the for membership1 */
}

```

```

if ((closest_mean < MINFLOAT) || (closest_mean > MAXFLOAT))
{
    return (FALSE);
}

/* printf("closest mean %f\n",closest_mean); */

Output_term_Deviation[output_lingui_no][membership] = (float) (fabs((double) (current_mean -
closest_mean)) / OUTPUT_OVERLAP);

/*
 * printf("Output term D
 * %8f\n",Output_term_Deviation[output_lingui_no][membership]);
 */
} /* End of the for membership */
return (TRUE);
} /* End of the function find_deviation */

```

```

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <conio.h>
#include "c:\borlandc\file\thesis.h"
#include "c:\borlandc\file\globalex.h"

```

```

/*****
 * This function update the Mean_input of the layer 2 and the Width of the
 * membership function
 *
 * Input parameter: lrate,
 * inputx.
 *
 * Output: TRUE if no error otherwise FALSE
 *
 */

```

```

int
error_backpro_layer2(float lrate, int input_index)

{
    unsigned int input_lingui_no,
        membership;

    float temp;

    for (input_lingui_no = 0; input_lingui_no < INPUT_LINGUI_NO; input_lingui_no++)
    {

        for (membership = 0; membership < Input_membership_array[input_lingui_no];
            membership++)
        {

            /* clrscr(); */

            /*
             * printf("\nMEMBERSHIP %d, INPUT_LINGUI_NO %d\n", membership,
             * input_lingui_no);
             */

            Change_E_respect_a = 0.0;

```

```

rule_connection(membership, input_lingui_no);

/*
 * printf("lrate %2.5f, index %d, inputx %2.5fn", lrate,
 * input_index, Inputx[input_lingui_no][input_index]);
 */

/*
 * printf("Mean_input %2.5f Diviation %2.5f C_E_respect_a %2.5f
 * act %2.5fn", Mean_input[input_lingui_no][membership],
 * Input_term_Deviation[input_lingui_no][membership],
 * Change_E_respect_a, (float)
 * l2_membership(Mean_input[input_lingui_no][membership],
 * Input_term_Deviation[input_lingui_no][membership],
 * Inputx[input_lingui_no][input_index]));
 */

/* Mean_input[input_lingui_no][membership] - */

Mean_input_curr[input_lingui_no][membership] += temp =
    (lrate * Change_E_respect_a *
    (float) l2_membership(Mean_input[input_lingui_no][membership],
        Input_term_Deviation[input_lingui_no][membership],
        Inputx[input_lingui_no][input_index]) *
    (2 * ((Inputx[input_lingui_no][input_index] - Mean_input[input_lingui_no][membership]) /
        (float) pow((double) Input_term_Deviation[input_lingui_no][membership], 2)
    )
    )
    );

/* printf("mean curr %2.5f ", temp); */

/* Input_term_Deviation[input_lingui_no][membership] - */

Input_term_Deviation_curr[input_lingui_no][membership] += temp =
    (lrate * Change_E_respect_a *
    (float) l2_membership(Mean_input[input_lingui_no][membership],
        Input_term_Deviation[input_lingui_no][membership],
        Inputx[input_lingui_no][input_index]) *
    (2 * ((float) pow((double) (Inputx[input_lingui_no][input_index] -
Mean_input[input_lingui_no][membership]), 2) /
        (float) pow((double) Input_term_Deviation[input_lingui_no][membership], 3)
    )
    )
    );

```

```

        /* printf("delta deviation %2.5f\n", temp); */
        /* getchar(); */
    }
}

return (TRUE);
}

/
*****
*****
* This function find all connection of rule nodes that are connected to a given
* in input term membership.
*
* Input parameter: input_term_membership,
* input_lingui_no.
*
* Output: TRUE if no error otherwise FALSE
*/

int
rule_connection(unsigned int input_term_membership, unsigned int input_lingui_no)
{
    unsigned int N,
        n,
        first_rule_node,
        next_rule_node;

    /*
    * printf("input_term_membership %d, input_lingui_no %d\n",
    * input_term_membership, input_lingui_no);
    */

    N = (Max_rule_no / (unsigned int) pow(((double) MEMBERSHIP_DEMEMSION,
        (double) (INPUT_LINGUI_NO - input_lingui_no))) - 1;

    first_rule_node = (input_term_membership * (unsigned int) pow(((double)
    MEMBERSHIP_DEMEMSION,
        (double) (INPUT_LINGUI_NO - input_lingui_no - 1)));

```

```

/* printf("\nFIRST_NODE %d\n",first_rule_node); */

membership_connection(first_rule_node, input_lingui_no, input_term_membership);

find_next_rule_node(first_rule_node, input_lingui_no, input_term_membership);


for (n = 1; n <= N; n++)
{
    next_rule_node = first_rule_node + (n * (unsigned int) pow(((double)
MEMBERSHIP_DEMEMSION,
(double) (INPUT_LINGUI_NO - input_lingui_no))));

    /* printf("\nNEXT_NODE %d\n", next_rule_node); */

    membership_connection(next_rule_node, input_lingui_no, input_term_membership);

    find_next_rule_node(next_rule_node, input_lingui_no, input_term_membership);

}

/* getchar(); */

return (TRUE);
}

/
*****
*****
* This function find all connection of input term membership that are connected to a given
* rule node.
*
* Input parameter: input_term_membership,
* input_node,
* rule_node.
*
* Output: TRUE if no error otherwise FALSE
*/

```

```

int
membership_connection(unsigned int rule_node, unsigned int input_node,
                     unsigned int input_term_membership)

{
    unsigned int i,
        is_minimum,
        input_lingui_no,
        Rule_no,
        membership;

    float minimum,
        temp_minimum;

    is_minimum = TRUE;
    Rule_no = rule_node;

    if (rule_node >= Max_rule_no || rule_node < 0)
    {
        printf("node greater expected value %d", rule_node);

        return (FALSE);
    }

    minimum = Act_l2[input_node][input_term_membership];

    /*
    * printf("input_node %d, input_term_membership %d\n", input_node,
    * input_term_membership);
    */

    for (i = INPUT_LINGUI_NO; i >= 1; i--)
    {
        input_lingui_no = INPUT_LINGUI_NO - i;

        membership = (rule_node / Rule_connect[i - 1]);

        /*
        * printf("input_lingui_no %d, membership %d ", input_lingui_no,
        * membership);
        */

        temp_minimum = Act_l2[input_lingui_no][membership];
    }
}

```

```

/*
 * printf("minimum %2.5f, temp_minimum %2.5f\n", minimum,
 * temp_minimum);
 */

if (minimum > temp_minimum)
{
    is_minimum = FALSE;

    /* printf("is_minimum is FALSE\n"); */

    break;
}

rule_node %= Rule_connect[i - 1];
/* printf("rule_node now %d\n", rule_node); */
}

if (is_minimum)
{
    /*
     * printf("R_no %d, E_respect_a %2.5f ", Rule_no,
     * Change_E_respect_a);
     */

    /* printf("E_sig_l3 %2.5f ", Error_signal_layer3[Rule_no]); */

    Change_E_respect_a += Error_signal_layer3[Rule_no];

    /* printf("Change_E_respect_a %2.5f\n", Change_E_respect_a); */
}

return (TRUE);
}

/
*****
*****
* This function find next number of rule nodes that are connected to a given
* in input term membership, and the current rule node that is calculate.
*

```



```

* Input parameter: input_term_membership,
* input_lingui_no,
*current_rule_node.
*
* Output: TRUE if no error otherwise FALSE
*/

```

```

int
find_next_rule_node(unsigned int next_node, unsigned int input_lingui_no,
                    unsigned int input_term_membership)

{
    unsigned int loop_node;

    loop_node = (((unsigned int) pow((double) MEMBERSHIP_DEMEMSION,
    (double) (INPUT_LINGUI_NO - input_lingui_no - 1))) - 1);

    /* printf(" loop node %d \n",loop_node); */

    while (loop_node > 0)
    {
        /* printf("\nNEXT_NODE %d \n", next_node + 1); */

        membership_connection(++next_node, input_lingui_no, input_term_membership);

        loop_node--;
    }

    return (TRUE);
}

```

```

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "c:\borlandc\file\thesis.h"
#include "c:\borlandc\file\globalex.h"

```

```

/
*****
*****
* This function calculate the Error signal for layer 3
*
* Input parameter: void
*
* Output: TRUE if no error otherwise FALSE
*
*/

int
error_backpro_layer3(void)
{
    unsigned int output_term_membership;

    int output_lingui_no,
        rule_no;

    float temp;

    /* printf("\nERROR SIGNAL LAYER 3\n"); */
    for (rule_no = 0; rule_no < RULE_NODE; rule_no++)
    {
        /* printf("RULE_NO %d\n",rule_no); */

        Error_signal_layer3[rule_no] = 0.0;

        for (output_lingui_no = 0; output_lingui_no < OUTPUT_LINGUI_NO; output_lingui_no++)
        {

            for (output_term_membership = 0; output_term_membership <
Output_membership_array[output_lingui_no]; output_term_membership++)
            {
                /* printf("op_t_mship %d, ", output_term_membership); */

                /*

```

```

* printf("W_l4 %2.5f, E_s_l4 %2.5f, ",
*,W_l4[output_lingui_no][output_term_membership][rule_no],
* Error_signal_layer4[output_lingui_no][output_term_membershi
* p]);
*/

if (W_l4[output_lingui_no][output_term_membership][rule_no] > 0.0)
{

    Error_signal_layer3[rule_no] += temp =
Error_signal_layer4[output_lingui_no][output_term_membership];

}

/*
* printf("E_s_l3 %2.5f, S_E %2.5f\n",temp,
* Error_signal_layer3[rule_no]);
*/
}          /* End of output_term_membership */

}          /* End of output_lingui_no */
/* getchar(); */

}          /* End of rule_no */
/* getchar(); */

return (TRUE);

}          /* End of error_backpro_layer3 */

```

```

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "c:\borlandc\file\thesis.h"
#include "c:\borlandc\file\globalex.h"

```

```

/
*****
*****
* This function calculate the Error signal for layer 4
*
* Input parameter: index
*
* Output: TRUE if no error otherwise FALSE
*
*/

int
error_backpro_layer4(float *Sum_width_act, float *Sum_mean_width_act)
{
    unsigned int output_lingui_no,
        i;

    float temp;

    /*
    * Update the Mean_output for each of the output_membership in layer 4
    */

    /* printf("\nNOW IN LAYER 4\n"); */

    for (output_lingui_no = 0; output_lingui_no < OUTPUT_LINGUI_NO; output_lingui_no++)
    {

        /* printf("Sum_w_act %2.5f ", Sum_width_act[output_lingui_no]); */

        /*
        * printf("Sum_m_w_act %2.5f\n",
        * Sum_mean_width_act[output_lingui_no]);
        */
    }
}

```

```

for (i = 0; i < Output_membership_array[output_lingui_no]; i++)
{
    /* printf("Mean_op %2.5f ", Mean_output[output_lingui_no][i]); */

    /*
    * printf("Out_Devi %2.5f ",
    * Output_term_Deviation[output_lingui_no][i]);
    */

    Error_signal_layer4[output_lingui_no][i] = temp = (Error_signal_layer5[output_lingui_no] *
        (
            (
                (
                    (Mean_output[output_lingui_no][i] *
                     Sum_width_act[output_lingui_no]
                    ) -
                    Sum_mean_width_act[output_lingui_no]
                ) *
                Output_term_Deviation[output_lingui_no][i]
            ) /
            pow(Sum_width_act[output_lingui_no], 2)
        )
    );

    /* printf("Error_s_l4 %2.5f\n", temp); */
}

/* End for i */

}

/* End for input_lingui_no */

return (TRUE);
}

/* End error_backpro_layrer4 */

```

```

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "c:\borlandc\file\thesis.h"
#include "c:\borlandc\file\globalex.h"

```

```

/*****
* This function calculate the Error of the network and update the Mean_output
* of the layer 5 and the Width of the membership function
*
* Input parameter: lrate,
* index.
*
* Output: TRUE if no error otherwise FALSE
*
*/

```

```

int
error_backpro_layer5(float lrate, unsigned int index)
{
    unsigned int output_lingui_no,
        center_index,
        devi_index,
        layer_5,
        i;

    float dev_mean,
        temp,
        sum_dev_mean,
        sum_dev_mean_width;

    /*
    * Update the Mean_output for each of the output_membership in layer 5
    */

    dev_mean = 0.0;
    /* printf("\nNOW IN LAYER 5 lrate %f, index %d\n",lrate, index); */

    for (output_lingui_no = 0; output_lingui_no < OUTPUT_LINGUI_NO; output_lingui_no++)
    {

        Sum_mean_width_act[output_lingui_no] = 0.0;
        Sum_width_act[output_lingui_no] = 0.0;
    }
}

```

```

sum_dev_mean = 0.0;
sum_dev_mean_width = 0.0;

for (i = 0; i < Output_membership_array[output_lingui_no]; i++)
{
    /* printf("Act_dup4 %f ", Act_downup_l4[output_lingui_no][i]); */

    /*
    * printf("Out_Devi %f ",
    * Output_term_Deviation[output_lingui_no][i]);
    */

    sum_dev_mean += temp = (Act_downup_l4[output_lingui_no][i] *
        Output_term_Deviation[output_lingui_no][i]);

    /* printf("sum_dev_mean %2.5f\n", temp); */
}
/* End for i */

/* printf("sum_dev_mean %2.5f\n", sum_dev_mean); */

/* getchar(); */

Sum_width_act[output_lingui_no] = sum_dev_mean;

for (center_index = 0; center_index < Output_membership_array[output_lingui_no];
    center_index++)
{
    dev_mean = (Act_downup_l4[output_lingui_no][center_index] *
        Output_term_Deviation[output_lingui_no][center_index]);

    /*
    * Calculate the Mean_output of the output layer but not update
    * untill all errors of each layer are calculate.
    */

    /* (Mean_output[output_lingui_no][center_index] + */
    Mean_output_delt[output_lingui_no][center_index] += temp =
        (Error_array[output_lingui_no] * lrate *
        (dev_mean / sum_dev_mean));

    /* printf("dev_mean %2.5f ", dev_mean); */
    /* printf("Mean_output_curr %2.5f ", temp); */
}

```

```

/*
 * printf("SM_output %2.5f\n",
 * Mean_output_delt[output_lingui_no][center_index]);
 */

}          /* End for center_index */

/* getchar(); */

for (i = 0; i < Output_membership_array[output_lingui_no]; i++)
{
    /* printf("Mean_op %f ", Mean_output[output_lingui_no][i]); */

    /*
     * printf("Out_Devi %f ",
     * Output_term_Deviation[output_lingui_no][i]);
     */

    /* printf("Act_dup4 %f ", Act_downup_l4[output_lingui_no][i]); */

    sum_dev_mean_width += temp = (Mean_output[output_lingui_no][i] *
                                   Act_downup_l4[output_lingui_no][i] *
                                   Output_term_Deviation[output_lingui_no][i]);

    /* printf("sum_d_m_w %2.5f\n ", temp); */
}          /* End for i */

Sum_mean_width_act[output_lingui_no] = sum_dev_mean_width;

/* printf("sum_d_m_w %2.5f\n", sum_dev_mean_width); */

/* getchar(); */

/* printf("Error_a %2.5f\n", Error_array[output_lingui_no]); */

for (devi_index = 0; devi_index < Output_membership_array[output_lingui_no];
     devi_index++)
{
    /*
     * Calculate the Output_term_Deviation of the output layer but
     * not update untill all errors of each layer are calculate.
     */

```



```

/* printf("devi_i %d ", devi_index); */

/*
* printf("Mean_op %2.5f ",
* Mean_output[output_lingui_no][devi_index]);
*/

/*
* printf("Act_dup4 %2.5f ",
* Act_downup_l4[output_lingui_no][devi_index]);
*/

/* Output_term_Deviation[output_lingui_no][devi_index] + */

Output_term_Deviation_curr[output_lingui_no][devi_index] += temp =
(
(Error_array[output_lingui_no] *
lrate
) *
(
(
(
(Mean_output[output_lingui_no][devi_index] *
sum_dev_mean
) - sum_dev_mean_width
) *
Act_downup_l4[output_lingui_no][devi_index]
)/
pow(sum_dev_mean, 2)
)
);

/*
* printf("Out_Devi %2.5f, S_out_D %2.5f\n", temp,
* Output_term_Deviation_curr[output_lingui_no][devi_index]);
*/

} /* End for devi_index */

Error_signal_layer5[output_lingui_no] = temp = Error_array[output_lingui_no];

/* getchar(); */

```

```

    }                /* End for input_lingui_no */

    return (TRUE);
}                /* End error_backpro_layrer5 */

/
*****
*****
* This function find all connection of rule nodes that are connected to a given
* in input term membership.
*
* Input parameter: input_term_membership,
* input_lingui_no.
*
* Output: TRUE if no error otherwise FALSE
*/

int
rule_connection(unsigned int input_term_membership, unsigned int input_lingui_no)
{
    unsigned int N,
        n,
        first_rule_node,
        next_rule_node;

    /*
    * printf("input_term_membership %d, input_lingui_no %d\n",
    * input_term_membership, input_lingui_no);
    */

    N = (Max_rule_no / (unsigned int) pow((double) MEMBERSHIP_DEMEMSION,
        (double) (INPUT_LINGUI_NO - input_lingui_no))) - 1;

    first_rule_node = (input_term_membership * (unsigned int) pow((double)
MEMBERSHIP_DEMEMSION,
        (double) (INPUT_LINGUI_NO - input_lingui_no - 1)));

    printf("\nfirst_node %d\n", first_rule_node);

    membership_connection(first_rule_node, input_lingui_no, input_term_membership);

    find_next_rule_node(first_rule_node, input_lingui_no, input_term_membership);

```

```

    for (n = 1; n <= N; n++)
    {
        next_rule_node = first_rule_node + (n * (unsigned int) pow(((double)
MEMBERSHIP_DEMEMSION,
        (double) (INPUT_LINGUI_NO - input_lingui_no))));

        printf("\nnext_node %d\n", next_rule_node);

        membership_connection(next_rule_node, input_lingui_no, input_term_membership);

        find_next_rule_node(next_rule_node, input_lingui_no, input_term_membership);

    }

    getchar();

    return (TRUE);
}

```

```

/
*****
*****
* This function find all connection of input term membership that are connected to a given
* rule node.
*
* Input parameter: input_term_membership,
* input_node,
* rule_node.
*
* Output: TRUE if no error otherwise FALSE
*/

```

```

int
membership_connection(unsigned int rule_node, unsigned int input_node,
    unsigned int input_term_membership)

{
    unsigned int i,
    is_minimum,

```

```

    input_lingui_no,
    Rule_no,
    membership;

float minimum,
    temp_minimum;

is_minimum = TRUE;
Rule_no = rule_node;

if (rule_node >= Max_rule_no || rule_node < 0)
{
    printf("node greater expected value %d", rule_node);

    return (FALSE);
}

minimum = Act_12[input_node][input_term_membership];

/*
* printf("input_node %d, input_term_membership %d\n", input_node,
* input_term_membership);
*/

for (i = INPUT_LINGUI_NO; i >= 1; i--)
{
    input_lingui_no = INPUT_LINGUI_NO - i;

    membership = (rule_node / Rule_connect[i - 1]);

    printf("input_lingui_no %d, membership %d\n", input_lingui_no, membership);

    temp_minimum = Act_12[input_lingui_no][membership];

    printf("minimum %2.5f, temp_minimum %2.5f\n", minimum, temp_minimum);

    if (minimum > temp_minimum)
    {
        is_minimum = FALSE;

        printf("is minimum is FALSE \n");

        break;
    }
}

```

```

        rule_node %= Rule_connect[i - 1];
        /* printf("rule_node now %d\n", rule_node); */
    }

    if (is_minimum)
    {
        printf("R_no %d, E_respect_a %2.5f ", Rule_no, Change_E_respect_a);

        printf("Error_signal_layer3 %2.5f ", Error_signal_layer3[Rule_no]);

        Change_E_respect_a += Error_signal_layer3[Rule_no];

        printf("E_respect_a %2.5f\n", Change_E_respect_a);
    }

    return (TRUE);
}

/
*****
*****
* This function find next number of rule nodes that are connected to a given
* in input term membership, and the current rule node that is calculate.
*
* Input parameter: input_term_membership,
* input_lingui_no,
*current_rule_node.
*
* Output: TRUE if no error otherwise FALSE
*/

int
find_next_rule_node(unsigned int next_node, unsigned int input_lingui_no,
                    unsigned int input_term_membership)
{
    unsigned int loop_node;

    loop_node = (((unsigned int) pow((double) MEMBERSHIP_DEMEMSION,
        (double) (INPUT_LINGUI_NO - input_lingui_no - 1))) - 1);

    /* printf(" loop node %d\n", loop_node); */

```

```
while (loop_node > 0)
{
    printf("\nnext node %d \n", next_node + 1);

    membership_connection(++next_node, input_lingui_no, input_term_membership);

    loop_node--;
}

return (TRUE);
}
```

```

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include "c:\borlandc\file\thesis.h"
#include "c:\borlandc\file\globalex.h"

```

```

/*****
* function updata_input_weight().
*
* This function updata the weight of the membership function which has the closest
* "mean" for a input data(inputx) of an input linguistic node identify as
* input_ling_index. The find_mini_ptr return the integer indicating the index
* of the membership function.
* This function call the call the function fi_mini_index() by referrencing the
* function address thus the find_mini_ptr is pointed to function fi_mini_index
*
* Input parameter:
* inputx,
* alpha,
* input_ling_index,
* (* find_mini_ptr)()
*
* Output: NONE
*
* Programmer:
* Leung Kam Lung 9/14/93MS. Thesis.
*
*/

```

```

void
update_input_l2_w(float inputx, float alpha, unsigned int input_ling_node)
{
    unsigned int mini_index;
    unsigned int (*find_mini_ptr) (float, unsigned int);

    find_mini_ptr = fi_l2_mini_index;
    mini_index = (*find_mini_ptr) (inputx, input_ling_node);

    /*
    * printf("input node %d mini_index %d before %f ",input_ling_node,
    * mini_index, Mean_input[input_ling_node][mini_index]);
    */
}

```

```

Mean_input[input_ling_node][mini_index] = (Mean_input[input_ling_node][mini_index] + (alpha *
    (inputx - Mean_input[input_ling_node][mini_index])));

/* printf("after %f\n",Mean_input[input_ling_node][mini_index]); */

}

/*****
* function fi_mini_index();
*
* This function find the shortest distance between the inputx and the "mean"
* of each of the membership function in a set of membership function which
* are belong to a particular input linguistic node.
*
* Input parameter:
* inputx,
* input_ling_node.
*
* Return value: mini_index
*
* programmer:Leung kam Lung9/16/93MS. Thesis.
*/

unsigned int
fi_l2_mini_index(float inputx, unsigned int input_ling_node)
{
    double minimum,
        temp,
        distance[MAX_MEMBERSHIP];

    unsigned int k,
        mini_index,
        membership;

    k = 0;
    temp = 0.0;
    mini_index = 0;
    membership = 0;
    minimum = 0.0;

    /* printf("input_ling_node %d\n",input_ling_node); */
    k = Input_membership_array[input_ling_node]; /* k contain the number of
        * membership function for
        * the input_ling_node */

```



```

for (membership = 0; membership < k; membership++)
{
    /*
    * calculates the distance between the inputx and the "meam" of each
    * membership function
    */

    distance[membership] = pow((double) (inputx - Mean_input[input_ling_node][membership]), 2);
}

/*
* Find the shortest distance between inputx and "meam" of all membership
* functions for a given input linguistic node.
*/

minimum = distance[0];
/* printf("minimum %f\n",minimum); */

for (membership = 1; membership < k; membership++)
{
    temp = distance[membership];
    /* printf("temp minimum %f\n",temp); */

    if (temp < minimum)
    {
        minimum = temp;
        /* printf("minimum %f\n",minimum); */

        mini_index = membership;

        /* printf(" minimum index %d\n",mini_index); */
    }
}

return (mini_index);
}

/*****
* function updata_output_w().
*
* This function updata the weight of the membership function which has the closest

```

```

* "mean" for a output data(inputy) of an output linguistic node identify as
* output_ling_index. The find_output_mini_ptr return the integer indicating the index
* of the membership function.
* This function call the call the function fi_out_mini_index() by referrencing the
* function address thus the find_output_mini_ptr is pointed to function fi_out_mini_index
*
* Input parameter:
*inputy,
*alpha,
*output_ling_index,
* (* find_output_mini_ptr)()
*
* Output: NONE
*
* Programmer:
* Leung Kam Lung 9/14/93MS. Thesis.
*
*/

```

```

void
update_output_w(float inputy, float alpha, unsigned int output_ling_node)
{
    unsigned int mini_index;
    unsigned int (*find_output_mini_ptr) (float, unsigned int);

    find_output_mini_ptr = fi_output_mini_index;
    mini_index = (*find_output_mini_ptr) (inputy, output_ling_node);

    /*
    * printf("output node %d mini_index %d before %f ",output_ling_node,
    * mini_index, Mean_output[output_ling_node][mini_index]);
    */

    Mean_output[output_ling_node][mini_index] = (Mean_output[output_ling_node][mini_index] +
        (alpha * (inputy - Mean_output[output_ling_node][mini_index])));

    /*
    * printf(" Mean_output =
    * %f\n",Mean_output[output_ling_node][mini_index]);
    */
    /* printf("after %f\n",Mean_output[output_ling_node][mini_index]); */
}

```

```

/*****
* function fi_output_mini_index();
*
* This function find the shortest distance between the inputy and the "mean"
* of each of the membership function in a set of membership function which
* are belong to a particular output linguistic node.
*
* Input parameter:
* inputy,
* output_ling_node.
*
* Return value: mini_index
*
* programmer:Leung kam Lung9/16/93MS. Thesis.
*/

```

```

unsigned int
fi_output_mini_index(float inputy, unsigned int output_ling_node)
{
    double minimum,
        temp,
        distance[MAX_MEMBERSHIP];

    unsigned int k,
        mini_index,
        membership;

    k = 0;
    temp = 0;
    mini_index = 0;
    membership = 0;
    minimum = 0.0;

    k = Output_membership_array[output_ling_node]; /* k contain the number
                                                    * of membership function
                                                    * for the
                                                    * input_ling_node */

    for (membership = 0; membership < k; membership++)
    {
        /*
        * calculates the distance between the inputx and the "meam" of each

```

```

    * membership function
    */
    distance[membership] = pow((double) (inputx - Mean_output[output_ling_node][membership]),
2);
}

/*
 * Find the shortest distance between inputx and "meam" of all membership
 * functions for a given input linguistic node.
 */

minimum = distance[0];
for (membership = 1; membership < k; membership++)
{
    temp = distance[membership];
    if (temp < minimum)
    {
        minimum = temp;
        mini_index = membership;
    }
}

return (mini_index);
}

```

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>
```

```
#include "c:\borlandc\file\thesis.h"
#include "c:\borlandc\file\globalex.h"
```

```
/
*****
*****
* This function get the mean value of all the membership function for all input linguistic
* nodes and all the membership function for all output linguistic nodes.
*
* Input parameter: Global paramters;
*Max_range, Min_range,
* Membership_array, Mean_input.
*
* Output:
*TRUE :If all mean values were calculated.
*False:If any mean values were not calculated.
*
* programmer:Leung Kam LungMS. Thesis9/28/93.
*
*
*All input and output variables are array pointer.
*
*/
```

```
unsigned int
get_means(void)
```

```
{
    unsigned int input_no,
                output_no,
                membership_index;

    clrscr();
    for (input_no = 0; input_no < INPUT_LINGUI_NO; input_no++)
    {
        if (find_mean(Max_range[input_no], Min_range[input_no],
                    Input_membership_array[input_no], input_no))
        {
            for (membership_index = 0; membership_index < Input_membership_array[input_no];
```

```

        membership_index++)
    {
        printf("Initail Mean_input%d = %f\n", membership_index,
Mean_input[input_no][membership_index]);

    }        /* End of for membership_index */

    }
    else
    {

        printf("fail to find input mean\n");
        return (FALSE);

    }
    getchar();
}        /* End of for input_no */

for (output_no = 0; output_no < OUTPUT_LINGUI_NO; output_no++)
{
    if (find_output_mean(Output_max_range[output_no], Output_min_range[output_no],
        Output_membership_array[output_no], output_no))
    {
        for (membership_index = 0; membership_index < Output_membership_array[output_no];
            membership_index++)
        {

            printf("Initiall Mean_output%d = %f\n", membership_index,
Mean_output[output_no][membership_index]);

        }        /* End for membership_index */
    }
    else
    {

        printf("fail to find output mean\n");
        return (FALSE);

    }
    getchar();
}        /* End of for output_no */

return (TRUE);

```

}

```

/*****
* This function calculate the mean value for input values of range from
* MAX_RANGE to MIN_RANGE. The mean value is start from left to right for
* a set of input term nodes which are correspond to a input linguistic
* node.
*
* input parameters
* MAX_RANGE
* MIN_RANGE
* NUM_MEMBERSHIP
* MEAN
*
* Output: TRUE if no arithmetic error
* FALSE otherwise.
*
* programmer: Leung Kam Lung MS. Thesis 9/28/93.
*/

```

```

unsigned int
find_mean(float max_range, float min_range, unsigned int num_membership, unsigned int
input_no)

```

```
{

```

```
    float segment;
    unsigned int i;

```

```
    /*

```

```
    * The value of segment descrypt equal distance between each mean.
    */

```

```
    printf("max_range %f , min_range %f, input_no\n", max_range, min_range, input_no);
    segment = (max_range - min_range) / (num_membership + 1);

```

```
    if ((segment > MAXFLOAT) || (segment < MINFLOAT))

```

```
    {
        return (FALSE);
    }

```

```
    getchar();

```

```
    /*

```

```

    * The value of i start from NUM_MEMBERSHIP and decrease to 1
    *
    */
    Mean_input[input_no][0] = min_range + segment;
    for (i = 1; i < num_membership; i++)
    {
        Mean_input[input_no][i] = Mean_input[input_no][i - 1] + segment;
    }
    return (TRUE);
}

/*****
* This function calculate the mean value for output values of range from
* MAX_RANGE to MIN_RANGE. The mean value is start from left to right for
* a set of output term nodes which are correspond to a output linguistic
* node.
*
* input parameters
* MAX_RANGE
* MIN_RANGE
* NUM_MEMBERSHIP
* MEAN
*
* Output: TRUE if no arithmetic error
* FALSE otherwise.
*
* programmer:Leung Kam LungMS. Thesis9/28/93.
*/

unsigned int
find_output_mean(float max_range, float min_range, unsigned int num_membership, unsigned int
output_no)
{
    float segment;
    unsigned int i;

    /*
    * The value of segment descrypt equal distance between each mean.
    */

    segment = (max_range - min_range) / (num_membership + 1);

```



```

/* printf("segment %f\n",segment); */

if ((segment > MAXFLOAT) || (segment < MINFLOAT))
{
    return (FALSE);
}

/*
 * The value of i start from NUM_MEMBERSHIP and decrease to 1
 */
Mean_output[output_no][0] = min_range + segment;

for (i = 1; i < num_membership; i++)
{
    Mean_output[output_no][i] = Mean_output[output_no][i - 1] + segment;
}
return (TRUE);
}

```

```
#include "c:\borlandc\file\thesis.h"
```

```
FILE *Inputx_fptr;  
FILE *Output_fptr;  
FILE *Input_term_Deviation_fptr;  
FILE *Final_input_term_deviation_fptr;  
FILE *Output_term_Deviation_fptr;  
FILE *Final_output_term_deviation_fptr;  
FILE *Mean_output_fptr;  
FILE *Final_mean_output_fptr;  
FILE *Mean_input_fptr;  
FILE *Final_mean_input_fptr;  
FILE *W_14_fptr;  
FILE *Error_fptr;  
FILE *Fback_fptr;  
FILE *Consigna_fptr;  
FILE *Slope_fptr;  
FILE *Max_data_fptr;
```

```
int Rule_connect[INPUT_LINGUI_NO];
```

```
/*  
 * These values are for filtering the input and output data.  
 */
```

```
float Max_input1,  
      Max_input2,  
      Max_output1;  
float Min_input1,  
      Min_input2,  
      Min_output1;  
float Input_slope1,  
      Input_slope2,  
      Control_signal,  
      Output_slope1;
```

```
float Max_range[INPUT_LINGUI_NO]; /* input range of linguistic nodes */  
float Min_range[INPUT_LINGUI_NO]; /* input range of linguistic nodes */  
float Output_max_range[OUTPUT_LINGUI_NO]; /* output range of linguistic nodes */  
float Output_min_range[OUTPUT_LINGUI_NO]; /* output range of linguistic nodes */  
float Input_term_Deviation[INPUT_LINGUI_NO][MAX_MEMBERSHIP]; /* input term deviation */  
float Input_term_Deviation_curr[INPUT_LINGUI_NO][MAX_MEMBERSHIP]; /* input term deviation */  
float Output_term_Deviation[OUTPUT_LINGUI_NO][MAX_MEMBERSHIP]; /* output term deviation */
```

```

float Output_term_Deviation_curr[OUTPUT_LINGUI_NO][MAX_MEMBERSHIP]; /* output term
deviation */
float Inputx[INPUT_LINGUI_NO][MAX_INPUT_INDEX]; /* training data */
float Output[OUTPUT_LINGUI_NO][MAX_INPUT_INDEX]; /* training data */
float Sum_width_act[OUTPUT_LINGUI_NO]; /* sum of width and activation of
* layer */
float Sum_mean_width_act[OUTPUT_LINGUI_NO]; /* sum of width and activation
* and mean of layer */
float Segment; /* number of segment of the range MAXIMIN
* ranger */

/*
* mean value of each membership of output term nodes
*/

float Mean_output[OUTPUT_LINGUI_NO][MAX_MEMBERSHIP];
float Mean_output_delt[OUTPUT_LINGUI_NO][MAX_MEMBERSHIP];

/*
* mean value of each membership of input term nodes
*/

float Mean_input[INPUT_LINGUI_NO][MAX_MEMBERSHIP];
float Mean_input_curr[INPUT_LINGUI_NO][MAX_MEMBERSHIP];

float Close_loop_input[INPUT_LINGUI_NO];

/* input number use for close loop simulation */
float Cal_error; /* calculate output error */

/* float Output; */

float Ref[MAX_TIME];

/*
* weight of input term nodes for all input linguistic nodes
*/

float W_l2[MAX_MEMBERSHIP][INPUT_LINGUI_NO];

/*
* activation value of layer 2 from input linguistic node 1 and node 2
*/

```

```

float Act_12[INPUT_LINGUI_NO][MAX_MEMBERSHIP];

/*
 * weight in layer 3
 */

float W_13[RULE_NODE][ALL_L2_MEMBERSHIP];

/*
 * activation value of each rule nodes in layer 3 from each input term nodes of layer 2
 */

float Act_13[RULE_NODE];

/*
 * error signal of layer 3, error for each of the rule node.
 */

float Error_signal_layer3[RULE_NODE];

float W_14[OUTPUT_LINGUI_NO][OUTPUT_TERM_MEMBERSHIP][RULE_NODE]; /* weight
in layer 4 */

/*
 * activation value of each output term nodes in layer 4 from each rule nodes of layer 3
 */

float Act_14[OUTPUT_LINGUI_NO][OUTPUT_TERM_MEMBERSHIP];

float Act_downup_14[OUTPUT_LINGUI_NO][OUTPUT_TERM_MEMBERSHIP];

float Output_up_down;
float Change_E_respect_a;

float Output_down_up[OUTPUT_LINGUI_NO][MAX_INPUT_INDEX];

int Max_rule_no;
int Before_loop;
int Max_indexes;

unsigned int INPUT_NODE;
unsigned int OUTPUT_NODE;

/*
 * contain the membership value of each of the input linguistic input node

```

*/

unsigned int Input_membership_array[INPUT_LINGUI_NO];

unsigned int Output_membership_array[OUTPUT_LINGUI_NO];

float Error_array[OUTPUT_LINGUI_NO][MAX_INPUT_INDEX];

float Error_signal_layer5[OUTPUT_LINGUI_NO];

float Error_signal_layer4[OUTPUT_LINGUI_NO][OUTPUT_TERM_MEMBERSHIP];

```

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include "c:\borlandc\file\thesis.h"
#include "c:\borlandc\file\globalex.h"

unsigned
initials_all(void)

{
    unsigned int output_lingui_no,
               center_index,
               rule_no,
               i,
               devi_index;

    init_maxmin_range();
    get_means();
    init_weight2();
    init_weight3();
    init_weight4();

    for (output_lingui_no = 0; output_lingui_no < OUTPUT_LINGUI_NO; output_lingui_no++)
    {
        for (center_index = 0; center_index < Output_membership_array[output_lingui_no];
             center_index++)
        {
            Mean_output_delt[output_lingui_no][center_index] = 0.0;
        }

        for (devi_index = 0; devi_index < Output_membership_array[output_lingui_no];
             devi_index++)
        {
            Output_term_Deviation_curr[output_lingui_no][devi_index] = 0.0;
        }
    }

    Rule_connect[0] = 1;
    Rule_connect[1] = Input_membership_array[0];

    for (i = 2; i < INPUT_LINGUI_NO; i++)
    {
        Rule_connect[i] = (Input_membership_array[i - 1] * Rule_connect[i - 1]);
    }
}

```

```

if (INPUT_LINGUI_NO > 1)
{
    Max_rule_no = Rule_connect[INPUT_LINGUI_NO - 1] *
Input_membership_array[INPUT_LINGUI_NO - 1];
}
printf(" Max rule no %d\n", Max_rule_no);
getchar();
return (TRUE);
}

```

```

#include "c:\borlandc\file\thesis.h"

float Max_range[INPUT_LINGUI_NO]; /* range of linguistic nodes */
float Min_range[INPUT_LINGUI_NO]; /* range of linguistic nodes */
float Segment; /* number of segment of the range MAXIMIN
               * ranger */

/*
 * mean value of each membership of output term nodes
 */

float Meam_output[OUTPUT_TERM_MEMBERSHIP];

/*
 * mean value of each membership of input term nodes
 */

float Mean_input[INPUT_LINGUI_NO][MAX_MEMBERSHIP];

/*
 * weight of input term nodes for all input linguistic nodes
 */

float W_l2[MAX_MEMBERSHIP][INPUT_LINGUI_NO];

/*
 * activation value of layer 2 from input linguistic node 1 and node 2
 */

float Act_l2[INPUT_LINGUI_NO][MAX_MEMBERSHIP];

/*
 * weight in layer 3
 */

float W_l3[RULE_NODE][ALL_L2_MEMBERSHIP];

/*
 * activation value of each rule nodes in layer 3 from each input term nodes of layer 2
 */

float Act_l3[RULE_NODE];

float W_l4[OUTPUT_TERM_NO][RULE_NODE]; /* weight in layer 4 */

/*

```



```

    * activation value of each output term nodes in layer 4 from each rule nodes of layer 3
    */

float Act_l4[OUTPUT_TERM_NO];

float Output_up_down;

float W_l5_output_up_down[OUTPUT_LINGUI_NO];

unsigned int INPUT_NODE;
unsigned int OUTPUT_NODE;

/*
    * contain the membership value of each of the input linguistic input node
    */

unsigned int Membership_array[INPUT_LINGUI_NO];

void
init_globe(void)
{
    Membership_array[0] = MEMBERSHIP_NO1;
    Membership_array[1] = MEMBERSHIP_NO2;
    Max_range[0] = 10.0;
    Min_range[0] = -10.0;
    Max_range[1] = 10.0;
    Min_range[1] = -10.0;
    NUM_MEMBERSHIP = 0;
}

void
init_weight(void)
{
    for (j = 0; j < Membership_array[input_lingui_no]; j++)
    {
        for (input_lingui_no = 0; input_lingui_no < INPUT_LINGUI_NO; input_lingui_no++)
        {
            W_l2[j][input_lingui_no] = random();
        }
    }
}

```

```

#include <time.h>
#include "c:\borlandc\file\thesis.h"
#include "c:\borlandc\file\globalex.h"

void
init_maxmin_range(void)

{
    unsigned int input_lingui_no,
        input_index,
        output_lingui_no,
        output_index;

    float temp,
        temp_min,
        temp_max;

    Input_membership_array[0] = MEMBERSHIP_NO1;
    Input_membership_array[1] = MEMBERSHIP_NO2;
    Output_membership_array[0] = OUTPUT_TERM_MEMBERSHIP;
    temp = 0;
    for (input_lingui_no = 0; input_lingui_no < INPUT_LINGUI_NO;
        input_lingui_no++)
    {

        temp_min = Inputx[input_lingui_no][0];
        temp_max = Inputx[input_lingui_no][0];

        for (input_index = 1; input_index < MAX_INPUT_INDEX; input_index++)
        {

            temp = Inputx[input_lingui_no][input_index];
            /* printf(" temp %f\n",temp); */

            if (temp < temp_min)
            {
                temp_min = temp;

                /*
                 * printf(" temp_min %f\n",temp_min); getchar();
                 */
            }
            else
            {

```

```

        if (temp > temp_max)
        {
            temp_max = temp;

            /*
             * printf(" temp_max %f\n",temp_max); getchar();
             */
        }
    }

    /*
     * printf("max %f, min %f, input_no %d\n", temp_max, temp_min,
     * input_lingui_no);
     */
    Max_range[input_lingui_no] = temp_max;
    Min_range[input_lingui_no] = temp_min;

    /*
     * printf(" max range of input_lingui_no %d is %f\n",input_lingui_no,
     * temp_max); printf(" min range of input_lingui_no %d is
     * %f\n",input_lingui_no, temp_min); getchar();
     */
    temp_min = 0;
    temp_max = 0;
}          /* End of input_lingui_no */

temp = 0;
for (output_lingui_no = 0; output_lingui_no < OUTPUT_LINGUI_NO;
    output_lingui_no++)
{
    temp_min = Output[output_lingui_no][0];
    temp_max = Output[output_lingui_no][0];
    for (output_index = 1; output_index < MAX_INPUT_INDEX; output_index++)
    {
        temp = Output[output_lingui_no][output_index];

        if (temp < temp_min)
        {
            temp_min = temp;
        }
    }
}

```

```

        else
        {
            if (temp > temp_max)
            {
                temp_max = temp;
            }
        }
    }
    /* End of output_index */

    Output_max_range[output_lingui_no] = temp_max;
    Output_min_range[output_lingui_no] = temp_min;

    /*
    * printf(" max range of output_lingui_no %d is
    * %f\n",output_lingui_no, temp_max); printf(" min range of
    * output_lingui_no %d is %f\n",output_lingui_no, temp_min);
    * getchar();
    */
    temp_min = 0;
    temp_max = 0;

}
/* End of output_lingui_no */

}

unsigned int
init_weight2(void)
{
    unsigned int input_lingui_no,
        membership_index;

    randomize();
    for (input_lingui_no = 0; input_lingui_no < INPUT_LINGUI_NO;
        input_lingui_no++)
    {
        for (membership_index = 0; membership_index < Input_membership_array[input_lingui_no];
            membership_index++)
        {
            W_l2[input_lingui_no][membership_index] = (random(10001) / 10000.0);

            /*
            * printf("input_lingui_no %d membership_index%d, weight =

```

```

        * %3.5f\n", input_lingui_no, membership_index,
        * W_12[input_lingui_no][membership_index]);
    */
    }
}
return (TRUE);
}

unsigned int
init_weight3(void)
{
    unsigned int rule_node_index,
        all_l2_membership_index;

    randomize();
    for (rule_node_index = 0; rule_node_index < RULE_NODE;
        rule_node_index++)
    {
        for (all_l2_membership_index = 0; all_l2_membership_index < ALL_L2_MEMBERSHIP;
            all_l2_membership_index++)
        {
            W_13[rule_node_index][all_l2_membership_index] = (random(10001) / 10000.0);

            /*
            * printf("rule_node %d all_l2_membership%d, weight = %3.5f\n",
            * rule_node_index, all_l2_membership_index,
            * W_13[rule_node_index][all_l2_membership_index]);
            */
        }
    }
    return (TRUE);
}

```

```

unsigned int
init_weight4(void)
{
    unsigned int output_term_node_index,
        rule_node_index,
        output_lingui_node;

```

```

for (output_lingui_node = 0; output_lingui_node < OUTPUT_LINGUI_NO; output_lingui_node++)
{
    for (output_term_node_index = 0; output_term_node_index < OUTPUT_TERM_MEMBERSHIP;
        output_term_node_index++)
    {
        for (rule_node_index = 0; rule_node_index < RULE_NODE;
            rule_node_index++)
        {
            W_l4[output_lingui_node][output_term_node_index][rule_node_index] = 1.0;

            /*
             * printf("output_term_node_index %d, rule_node %d, weight =
             * %3.5f\n", output_term_node_index, rule_node_index,
             * W_l4[output_term_node_index][rule_node_index]);
             */
        }
    }
}
return (TRUE);
}

```

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
```

```
#include "c:\borlandc\file\thesis.h"
#include "c:\borlandc\file\globalex.h"
```

```
int
get_act2(void)
```

```
{
    int retval;
    unsigned int input_lingui_no;
    unsigned int input_index;

    int (*act_function_ptr) (float, unsigned int);

    act_function_ptr = activation_l2;

    for (input_lingui_no = 0; input_lingui_no < INPUT_LINGUI_NO; input_lingui_no++)
    {
        for (input_index = 0; input_index < MAX_INPUT_INDEX; input_index++)
        {
            retval = (*act_function_ptr) (Inputx[input_lingui_no][input_index], input_lingui_no);

            if (retval == FALSE)
            {
                printf("return value of act_function_ptr is > or < max,min double\n");
                return (FALSE);
            }
            /* End of if retval */
        }
        /* End of for input_index */
    }
    /* End of for input_lingui_no */
    return (TRUE);
}
```

```
int
activation_l2(float inputx, unsigned int input_lingui_no)
```

```
{
    int membership;
```

```

float temp;

double (*activation_l2_ptr) (float, float, float);

activation_l2_ptr = l2_membership;

for (membership = 0; membership < Input_membership_array[input_lingui_no]; membership++)
{
    temp = Act_l2[input_lingui_no][membership] = (float) (*activation_l2_ptr)
        (Mean_input[input_lingui_no][membership],
        Input_term_Deviation[input_lingui_no][membership],
        inputx);

    if ((temp > MAXFLOAT) || (temp < MINFLOAT))
    {
        printf("Act_l2 value of input_lingui_no %d membership %d is > or < expected value\n",
            input_lingui_no, membership);

        return (FALSE);
    }

}

/* End of for membership */

return (TRUE);

}

/* End of activation_l2 */

```

```

/*****
* The function calculates the activation value for membership function in
* layer 2 of the NNFLC.
* input parameters:
* mean
* deviation
* u(input value)
* output:
* return type double
*
*****/

```


*/

double

l2_membership(float mean, float deviation, float u)

{

return (exp(-pow(((double) u - (double) mean), 2) /
(2 * pow((double) deviation, 2))) /
(2.50662825 * (double) deviation));

}

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
```

```
#include "c:\borlandc\file\thesis.h"
#include "c:\borlandc\file\globalex.h"
```

```
/******
```

```
 * function get_act2()
```

```
 *
```

```
 * This function calculates the output of the layer two of the network. This
 * function uses a function pointers points to function activation_l2.
```

```
 *
```

```
 * Input parameter: none.
```

```
 *
```

```
 * Return value: int type.
```

```
 *
```

```
 *programmer:Leung Kam LungMS. Thesis9/28/93
```

```
 *
```

```
 */
```

```
int
```

```
get_act2(unsigned int input_lingui_no, unsigned int input_index)
```

```
{
```

```
    int retval;
```

```
    int (*act_function_ptr) (float, unsigned int);
```

```
    act_function_ptr = activation_l2;
```

```
    retval = (*act_function_ptr) (Inputx[input_lingui_no][input_index], input_lingui_no);
```

```
    if (retval == FALSE)
```

```
    {
```

```
        printf("return value of act_function_ptr is > or < max,min double\n");
```

```
        return (FALSE);
```

```
    } /* End of if retval */
```

```
    return (TRUE);
```

```
}
```

```
/
```

```
*****
```

```
*****
```

```
 * This function is called by get_act2 to set up the parameters for calculates the output value
```

```

* for layer of the network.
*
* Input parameter: inputx,
*input_lingui_no.
*
* Return value: TRUE if successfull calculate the output values.
* FALSE otherwise.
*
*
*programmer:Leung Kam LungMS. Thesis9/28/93
*/

int
activation_l2(float inputx, unsigned int input_lingui_no)
{
    int membership;
    float temp;

    double (*activation_l2_ptr) (float, float, float);

    activation_l2_ptr = l2_membership;

    for (membership = 0; membership < Input_membership_array[input_lingui_no]; membership++)
    {
        temp = Act_l2[input_lingui_no][membership] = (float) (*activation_l2_ptr)
            (Mean_input[input_lingui_no][membership],
            Input_term_Deviation[input_lingui_no][membership],
            inputx);

        /*
        * printf("act layer2 %f\n",temp); getchar();
        */

        if (temp >= MAXFLOAT)
        {
            printf("Act_l2 value of input_lingui_no %d membership %d is > or < expected value\n",
                input_lingui_no, membership);

            return (FALSE);
        }
    }
}

```

```

    }                /* End of for membership */

    return (TRUE);

}                /* End of activation_l2 */

```

```

/*****
* The function calculates the activation value for membership function in
* layer 2 of the NNFLC.
* input parameters:
* mean
* deviation
* u(input value)
* output:
* Return type double
*
* programmer: Leung Kam Lung MS. Thesis 9/28/93
*
*/

```

```

double
l2_membership(float mean, float deviation, float u)
{
    /*
    * printf("2 deviation %f mean %f float %f", deviation, mean, u);
    * getchar();
    */

    return (exp(-pow(((double) u - (double) mean), 2) / (pow((double) deviation, 2))));
}

```

```

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>

#include "c:\borlandc\file\thesis.h"
#include "c:\borlandc\file\globalex.h"

/
*****
*****
* function get_act3()
*
* This function find the activation value of the layer three by perform the minimum of
* the output of the input term nodes which are the output of the membership in layer 2.
* The activation value of this layer(rule_nodes) are the minimum value from the output
* of the layer 2.
*
* Input parameter: input_lingui_no1,
*                  input_lingui_no2.
*
*
* Return value: TRUE if successfull calculate the output values.
* FALSE otherwise.
*
*
*programmer:Leung Kam LungMS. Thesis9/28/93
*
*/

int
get_act3(unsigned int input_lingui_no1, unsigned int input_lingui_no2)
{
    unsigned int rule_node;

    unsigned int lingno1_membership;
    unsigned int lingno2_membership;

    float smallest;
    float smallest2;

    rule_node = 0;
    for (lingno1_membership = 0; lingno1_membership < MEMBERSHIP_NO1;
        lingno1_membership++)
    {

```

```

smallest = Act_l2[input_lingui_no1][lingno1_membership];

for (lingno2_membership = 0; lingno2_membership < MEMBERSHIP_NO2;
lingno2_membership++)
{
    /* printf("rule_node = %d\n", rule_node); */
    smallest2 = Act_l2[input_lingui_no2][lingno2_membership];
    /* printf("smallest = %f\n", smallest); */
    /* printf("smallest2 = %f\n", smallest2); */

    if (smallest < smallest2)
    {
        Act_l3[rule_node] = smallest;
        /* printf("Act_l3 = %f\n", Act_l3[rule_node]); */
    }
    else
    {
        Act_l3[rule_node] = smallest2;
        /* printf("Act_l3 = %f\n", Act_l3[rule_node]); */
    }

    rule_node++;
    if (rule_node > RULE_NODE)
    {
        printf("ERROR number of rule nodes are greater than the expected value\n");
        return (FALSE);
    }
    /* getch(); */
}
/* End of lingno2_membership */
}
/* End of lingno1_membership */

return (TRUE);
}
/* End of the program */

```

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
```

```
#include "c:\borlandc\file\thesis.h"
#include "c:\borlandc\file\globalex.h"
```

```
/******
```

```
 * function get_act4()
```

```
 *
```

```
 * This function calculates the output of the layer two of the network. This
 * function uses a function pointers points to function activation_l2.
```

```
 *
```

```
 * Input parameter: none.
```

```
 *
```

```
 * Return value: int type.
```

```
 *
```

```
 * programmer: Leung Kam Lung MS. Thesis 9/28/93
```

```
 *
```

```
 */
```

```
int
```

```
get_act4(unsigned int output_lingui_no, unsigned int index)
```

```
{
```

```
    int retval;
```

```
    int (*act_function_ptr) (float, unsigned int);
```

```
    act_function_ptr = activation_l4;
```

```
    retval = (*act_function_ptr) (Output[output_lingui_no][index], output_lingui_no);
```

```
    if (retval == FALSE)
```

```
    {
```

```
        printf("return value of act_function_ptr is > or < max,min double\n");
```

```
        return (FALSE);
```

```
    } /* End of if retval */
```

```
    return (TRUE);
```

```
}
```

```
/
```

```
*****
*****
```

```
 * This function is called by get_act2 to set up the parameters for calculates the output value
```

```

* for layer of the network.
*
* Input parameter: inputx,
*input_lingui_no.
*
* Return value: TRUE if successfull calculate the output values.
* FALSE otherwise.
*
*
*programmer:Leung Kam LungMS. Thesis9/28/93
*/

int
activation_l4(float output, unsigned int output_lingui_no)

{
    int membership;
    float temp4;

    double (*activation_l4_ptr) (float, float, float);

    activation_l4_ptr = l4_membership;

    for (membership = 0; membership < Output_membership_array[output_lingui_no]; membership++)
    {

        temp4 = Act_l4[output_lingui_no][membership] = (float) (*activation_l4_ptr)
            (Mean_output[output_lingui_no][membership],
            Output_term_Deviation[output_lingui_no][membership],
            output);

        /* printf("act4 %f\n",temp4); */
        if (temp4 > MAXFLOAT)
        {

            printf("Act_l4 value of output_lingui_no %d membership %d is > or < expected value\n",
                output_lingui_no, membership);

            return (FALSE);
        }

    }

    /* End of for membership */

    return (TRUE);
}

```



```

}          /* End of activation_l2 */

```

```

/*****
* The function calculates the activation value for membership function in
* layer 4 of the NNFLC.
* input parameters:
* mean
* deviation
* u(input value)
* output:
* Return type double
*
* programmer: Leung Kam Lung MS. Thesis 9/28/93
*
*/

```

```

double
l4_membership(float mean, float deviation, float u)
{
    /*
    * printf("4 deviation %f, mean %f, float %f", deviation, mean, u);
    * getchar();
    */
    return (exp(-pow(((double) u - (double) mean), 2) / (pow((double) deviation, 2))));
}

```

```

#include <stdio.h>
#include <stdlib.h>
#include <stdio.h>
#include "c:\borlandc\file\thesis.h"
#include "c:\borlandc\file\globalex.h"

```

```

int
find_max_w4(void)

```

```

{
    float weight4,
        max_w4;
    unsigned int output_lingui_no,
        output_term_membership,
        rule_index,
        max_w4_index;

    for (output_lingui_no = 0; output_lingui_no < OUTPUT_LINGUI_NO; output_lingui_no++)
    {

        for (rule_index = 0; rule_index < RULE_NODE; rule_index++)
        {

            /*
             * printf("output_term_membership %d, rule
             * %d",output_term_membership, rule_index);
             */

            /*
             * printf(" weight4 %f
             * ",W_14[output_lingui_no][output_term_membership][rule_index]);
             */

            max_w4 = -99999.0;

            for (output_term_membership = 0; output_term_membership <
Output_membership_array[output_lingui_no];
                output_term_membership++)
            {

                weight4 = W_14[output_lingui_no][output_term_membership][rule_index];

```

```

/* printf("weight4 %f\n",weight4); */
/*
if (weight4 >= max_w4)
{
    max_w4 = weight4;
    max_w4_index = output_term_membership;

    /*
    * printf(" max_w4_index %d, w_4 %f\n ",max_w4_index,
    * weight4);
    */
}

if (weight4 > MAXFLOAT)
{

    printf("weight4 value of output_lingui_no %d output term membership %d rule_index &d
is > or < expected value\n",
        output_lingui_no, output_term_membership, rule_index);

    return (FALSE);

}

} /* End for output_term_membership */

for (output_term_membership = 0; output_term_membership <
Output_membership_array[output_lingui_no];
    output_term_membership++)
{

    if (output_term_membership == max_w4_index)
    {

        W_l4[output_lingui_no][output_term_membership][rule_index] = 1.0;

    }
    else
    {

        W_l4[output_lingui_no][output_term_membership][rule_index] = 0.0;

    }
}

```

```

        } /* End second for output_term_membership */
    } /* End rule_index */
} /* End for output_lingui_no */
return (TRUE);
} /* End of function change_w4 */

```

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
```

```
#include "c:\borlandc\file\thesis.h"
#include "c:\borlandc\file\globalex.h"
```

```

/*****
 * function find_output()
 *
 * This function calculates the output of the layer two of the network. This
 * function uses a function pointers points to function activation_l2.
 *
 * Input parameter: none.
 *
 * Return value: int type.
 *
 * programmer: Leung Kam Lung MS. Thesis 9/28/93
 */

```

```

int
find_output(void)
{
    int retval2,
        retval3,
        retval4;

    unsigned int input_lingui_no,
        input_lingui_no1,
        input_lingui_no2,
        output_lingui_no;

    float out1,
        f,
        f2;

    unsigned int index,
        output_term_membership,
        membership;

    int (*act_function_ptr2) (unsigned int, unsigned int);
    int (*act_function_ptr3) (unsigned int, unsigned int);
    int (*act_function_downup_ptr4) (void);

```

```

input_lingui_no1 = 0;
input_lingui_no2 = 1;
output_lingui_no = 0;

act_function_downup_ptr4 = get_downup_act4;
act_function_ptr3 = get_act3;
act_function_ptr2 = get_act2;

for (index = 0; index < MAX_INPUT_INDEX; index++)
{
    for (input_lingui_no = 0; input_lingui_no < INPUT_LINGUI_NO; input_lingui_no++) .
    {
        retval2 = (*act_function_ptr2) (input_lingui_no, index);

        for (membership = 0; membership < Input_membership_array[input_lingui_no]; member-
ship++)
        {
            fprintf(Act2_fptr, "%1.6f ", Act_l2[input_lingui_no][membership]);
        }

        } /* End of for input_lingui_no */
        fprintf(Act2_fptr, "\n");

        retval3 = (*act_function_ptr3) (input_lingui_no1, input_lingui_no2);
        for (membership = 0; membership < RULE_NODE; membership++)
        {
            fprintf(Act3_fptr, "%1.6f ", Act_l3[membership]);
        }
        fprintf(Act3_fptr, "\n");

        retval4 = (*act_function_downup_ptr4) ();
        for (output_lingui_no = 0; output_lingui_no < OUTPUT_LINGUI_NO; output_lingui_no++)
        {
            for (membership = 0; membership < Output_membership_array[output_lingui_no]; member-
ship++)
            {
                fprintf(Act4_fptr, "%1.6f ", Act_downup_l4[output_lingui_no][membership]);
            }

        }
        fprintf(Act4_fptr, "\n");
    }
}

```

```

for (output_lingui_no = 0; output_lingui_no < OUTPUT_LINGUI_NO; output_lingui_no++)
{
    output_down_up[index][output_lingui_no] = 0.0;
    f = f2 = 0.0;
    for (output_term_membership = 0; output_term_membership <
Output_membership_array[output_lingui_no];
        output_term_membership++)
    {
        f += (Mean_output[output_lingui_no][output_term_membership] *
            Output_term_Deviation[output_lingui_no][output_term_membership] *
            Act_downup_14[output_lingui_no][output_term_membership]);

        f2 += (Output_term_Deviation[output_lingui_no][output_term_membership] *
            Act_downup_14[output_lingui_no][output_term_membership]);
    }
    /* End of output_term_membership */

    out1 = output_down_up[index][output_lingui_no] = f / f2;

    /*
    * printf(" actual output %f ", Output[0][index]); printf(" the
    * calculated output is %f\n", out1); getchar();
    */
}
/* End of output_lingui_no */

if (retval2 == FALSE)
{
    printf("return value of act_function_ptr2 is > or < max,min double\n");
    return (FALSE);
}
/* End of if retval */

if (retval3 == FALSE)
{
    printf("return value of act_function_ptr3 is > or < max,min double\n");
    return (FALSE);
}
/* End of if retval */

if (retval4 == FALSE)
{
    printf("return value of act_function_downup_ptr4 is > or < max,min double\n");
    return (FALSE);
}
/* End of if retval */

```

```
    }                /* End of for input_index */  
  
    fclose(Act2_fptr);  
    fclose(Act3_fptr);  
    fclose(Act4_fptr);  
  
    return (TRUE);  
}
```



```
#include <stdio.h>
#include <stdlib.h>
#include "c:\borlandc\file\thesis.h"
#include "c:\borlandc\file\globalex.h"
```

```
/******
```

```
* function init_phase1()
*
```

```
* This function excute all initial procedure for getting the center of the
* membership and the deviation of input and output term nodes. So after this
* function the input signal can be pass through the membership on layer3 and
* layer4.
*
```

```
*Input parameter: inputx1,
*inputx2.
*
```

```
* Return value: TRUE if each of the center of membership and the deviation
*are calculated.
*FALSE otherwise.
*
```

```
* Programmer:
* Leung Kam Lung 9/30/93MS. Thesis.
*
```

```
*/
```

```
int
init_phase1(void)
```

```
{
    int i,
        ind;
    float time,
        time2;
    unsigned int input_lingui_no,
        output_lingui_no;

    ind = 1;
    time = 0.10;
    time2 = 0.10;
    initials_all();
    input_lingui_no = 0;
    output_lingui_no = 0;
    while ((time > 0.000001) || (time2 > 0.000001))
    {
        for (i = 0; i < MAX_INPUT_INDEX; i++)
```

```

    {
        update_input_l2_w(Inputx[input_lingui_no][i], time, input_lingui_no);
        update_input_l2_w(Inputx[input_lingui_no + 1][i], time, input_lingui_no + 1);
        update_output_w(Output[output_lingui_no][i], time2, output_lingui_no);
    }
    time = time * 0.95;
    time2 = time2 * 0.95;

    ind++;
}

printf(" ind %d\n", ind);

if (!save_mean_input())
{
    printf("Mean_input FALSE\n");
}

if (!save_mean_output())
{
    printf("Mean_output FALSE\n");
}

/*
 * up to now only the center(mean) of the input and output membership are
 * calculated
 */

for (input_lingui_no = 0; input_lingui_no < INPUT_LINGUI_NO; input_lingui_no++)
{
    if (find_input_deviation(input_lingui_no))
    {
        printf(" Deviation of each Membership is find for INPUT_TERM_NODE %d\n",
input_lingui_no);
    }
    else
    {
        printf(" NOT Deviation of each Membership is find for INPUT_TERM_NODE %d\n",
input_lingui_no);
        return (FALSE);
    }
}

```

```

    }          /* End for input_lingio_no */

    if (!save_input_deviation())
    {
        printf("CAN NOT SAVE INPUT DEVIATION\n");
    }

    for (output_lingui_no = 0; output_lingui_no < OUTPUT_LINGUI_NO; output_lingui_no++)
    {

        if (find_output_deviation(output_lingui_no))
        {
            printf(" Deviation of each Membership is find for OUTPUT_TERM_NODE %d\n",
output_lingui_no);
        }
        else
        {
            printf(" NOT Deviation of each Membership is find for OUTPUT_TERM_NODE %d\n",
output_lingui_no);
            return (FALSE);
        }

    }          /* End for output_lingio_no */

    if (!save_output_deviation())
    {
        printf("CAN NOT SAVE OUTPUT DEVIATION\n");
    }

    /*
    * now all deviation of each membership for input term nodes and output
    * term nodes are calculated
    */

    if (!update_w_14())
    {
        printf("update_w_14 FALSE\n");
        return (FALSE);
    }

    if (!save_w4())
    {
        printf("can not save weight 4\n");
    }

```

```
        return (FALSE);
    }

    if (!find_output())
    {
        printf(" Can not find the output\n");
        return (FALSE);
    }

    return (TRUE);
}
```

```

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#include "c:\borlandc\file\thesis.h"
#include "c:\borlandc\file\globalex.h"

int
save_w4(void)

{
    float weight4;
    unsigned int output_lingui_no;
    unsigned int output_term_membership;
    unsigned int rule_index;

    for (output_lingui_no = 0; output_lingui_no < OUTPUT_LINGUI_NO; output_lingui_no++)
    {

        for (output_term_membership = 0; output_term_membership <
Output_membership_array[output_lingui_no];
            output_term_membership++)
        {

            for (rule_index = 0; rule_index < RULE_NODE; rule_index++)
            {

                fprintf(W_14_fptr, "%1.6f ",
W_14[output_lingui_no][output_term_membership][rule_index]);

                } /* End rule_index */
                fprintf(W_14_fptr, "\n");
            } /* End for output_term_membership */
        } /* End for output_lingui_no */
        fclose(W_14_fptr);
        return (TRUE);
    } /* End of function change_w4 */
}

```

```

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#include "c:\borlandc\file\thesis.h"
#include "c:\borlandc\file\globalex.h"

int
save_w4(void)
{
    float weight4;
    unsigned int output_lingui_no;
    unsigned int output_term_membership;
    unsigned int rule_index;

    for (output_lingui_no = 0; output_lingui_no < OUTPUT_LINGUI_NO; output_lingui_no++)
    {
        for (output_term_membership = 0; output_term_membership <
Output_membership_array[output_lingui_no];
output_term_membership++)
        {
            for (rule_index = 0; rule_index < RULE_NODE; rule_index++)
            {
                fprintf(W_14_fptr, "%d ", (int)
W_14[output_lingui_no][output_term_membership][rule_index]);

                } /* End rule_index */
                fprintf(W_14_fptr, "\n");
            } /* End for output_term_membership */
        } /* End for output_lingui_no */

        if (fclose(W_14_fptr))
        {
            return (FALSE);
        }
        return (TRUE);
    } /* End of function change_w4 */

```

```

#include <stdio.h>
#include <stdlib.h>
#include "c:\borlandc\file\thesis.h"
#include "c:\borlandc\file\globalex.h"

```

```
int
```

```
update_w(int prt_index)
```

```

{
    unsigned int center_index,
        devi_index,
        input_lingui_no,
        output_lingui_no,
        membership;
    int go_print;

    if (prt_index == 249)
    {
        go_print = TRUE;
    }
    else
    {
        go_print = FALSE;
    }

    for (output_lingui_no = 0; output_lingui_no < OUTPUT_LINGUI_NO; output_lingui_no++)
    {
        for (center_index = 0; center_index < Output_membership_array[output_lingui_no];
            center_index++)
        {
            Mean_output[output_lingui_no][center_index] = Mean_output[output_lingui_no][center_index]
+
            Mean_output_delt[output_lingui_no][center_index];

            if (go_print == TRUE)
            {
                fprintf(Final_mean_output_fptr, "%2.6f ", Mean_output[output_lingui_no][center_index]);
            }

            Mean_output_delt[output_lingui_no][center_index] = 0.0;
        }
        /* center_index */
    }
}

```

```

if (go_print == TRUE)
{
    fprintf(Final_mean_output_fptr, "\n");
}

for (devi_index = 0; devi_index < Output_membership_array[output_lingui_no];
    devi_index++)
{
    Output_term_Deviation[output_lingui_no][devi_index] =
Output_term_Deviation[output_lingui_no][devi_index] +
    Output_term_Deviation_curr[output_lingui_no][devi_index];

    if (go_print == TRUE)
    {
        fprintf(Final_output_term_deviation_fptr, "%1.6f ",
Output_term_Deviation[output_lingui_no][devi_index]);
    }

    Output_term_Deviation_curr[output_lingui_no][devi_index] = 0.0;
}
/* End for devi_index */
if (go_print == TRUE)
{
    fprintf(Final_output_term_deviation_fptr, "\n");
}
}
/* End for output_lingui_no */

for (input_lingui_no = 0; input_lingui_no < INPUT_LINGUI_NO; input_lingui_no++)
{
    for (membership = 0; membership < Input_membership_array[input_lingui_no];
        membership++)
    {
        /*
        * printf("Mean_input no %d mem_ship %d %2.5f
        * ",input_lingui_no,membership,
        * Mean_input[input_lingui_no][membership]);
        */

        Mean_input[input_lingui_no][membership] = Mean_input[input_lingui_no][membership] +
            Mean_input_curr[input_lingui_no][membership];
    }
}

```



```

        if (go_print == TRUE)
        {
            fprintf(Final_mean_input_fptr, "%2.6f ", Mean_input[input_lingui_no][membership]);
        }

        /*
        * printf("delta %2.5f ",
        * Mean_input_curr[input_lingui_no][membership]);
        */

        /*
        * printf("FMean_input %2.5f\n",
        * Mean_input[input_lingui_no][membership]);
        */

        Mean_input_curr[input_lingui_no][membership] = 0.0;

        Input_term_Deviation[input_lingui_no][membership] =
        Input_term_Deviation[input_lingui_no][membership] +
        Input_term_Deviation_curr[input_lingui_no][membership];

        if (go_print == TRUE)
        {
            fprintf(Final_input_term_deviation_fptr, "%1.6f ",
            Input_term_Deviation[input_lingui_no][membership]);
        }

        /*
        * printf("delta Mean_Devi input_l_no %d mem_ship %d
        * %2.5f\n",input_lingui_no,membership,
        * Input_term_Deviation_curr[input_lingui_no][membership]);
        */
        Input_term_Deviation_curr[input_lingui_no][membership] = 0.0;

    }          /* End for membership */

}          /* End for input_lingui_no */
if (go_print == TRUE)
{
    fprintf(Final_mean_input_fptr, "\n");
    fprintf(Final_input_term_deviation_fptr, "\n");
}

return (TRUE);

```

} /* End of update_w */

```

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "c:\borlandc\file\thesis.h"
#include "c:\borlandc\file\globalex.h"

```

```

main(void)

```

```

{

```

```

    unsigned int index,
        input_index,
        find,
        counter,
        total;

```

```

    float input1,
        input2,
        output1,
        max_input1,
        max_input2,
        max_output1,
        min_input1,
        min_input2,
        min_output1,
        input_slope1,
        input_slope2,
        output_slope1;

```

```

    float number;
    long big,
        small;

```

```

    float array_input1[MAX_INPUT_INDEX],
        array_input2[MAX_INPUT_INDEX],
        array_output1[MAX_INPUT_INDEX];

```

```

    float array_counter[MAX_INPUT_INDEX];

```

```

    FILE *Error_in;
    FILE *Fback_in;
    FILE *Consigna_out;
    FILE *Max_input_value_fptr;
    FILE *Slope_fptr;

```

```

if ((Error_in = fopen("error.dat", "r")) == NULL)
{
    fprintf(stderr, "CAN NOT OPEN ERROR FILE\n");
    return (FALSE);
}

if ((Fback_in = fopen("fback.dat", "r")) == NULL)
{
    fprintf(stderr, "CAN NOT OPEN FBACK FILE\n");
    return (FALSE);
}

if ((Consigna_out = fopen("consigna.dat", "r")) == NULL)
{
    fprintf(stderr, "CAN NOT OPEN CONSIGNA FILE\n");
    return (FALSE);
}

for (index = 0; index < MAX_INPUT_INDEX; index++)
{
    array_counter[index] = -10.0;
}

total = 0;

for (index = 0; index < MAX_INPUT_INDEX; index++)
{
    fscanf(Error_in, "%f", &input1);
    fscanf(Fback_in, "%f", &input2);
    fscanf(Consigna_out, "%f", &output1);

    do
    {
        find = FALSE;
        counter = random(MAX_INPUT_INDEX);
        if (array_counter[counter] == -10.0)
        {
            array_counter[counter] = 10.0;
            array_input1[counter] = input1;
            array_input2[counter] = input2;
            array_output1[counter] = output1;
        }
    }
}

```

```

        find = TRUE;
        total++;
    }
} while (find == FALSE);

}

/*****
* This next two for loop is to normalize the training data to a range
* of -1 and 1.
*
*/

max_input1 = array_input1[0];
max_input2 = array_input2[0];
max_output1 = array_output1[0];

min_input1 = array_input1[0];
min_input2 = array_input2[0];
min_output1 = array_output1[0];

for (index = 1; index < MAX_INPUT_INDEX; index++)
{
    if (max_input1 < array_input1[index])
    {
        max_input1 = array_input1[index];
    }

    if (min_input1 > array_input1[index])
    {
        min_input1 = array_input1[index];
    }

    if (max_input2 < array_input2[index])
    {
        max_input2 = array_input2[index];
    }

    if (min_input2 > array_input2[index])
    {
        min_input2 = array_input2[index];
    }
}

```

```

        if (max_output1 < array_output1[index])
        {
            max_output1 = array_output1[index];
        }

        if (min_output1 > array_output1[index])
        {
            min_output1 = array_output1[index];
        }

    }          /* End for index loop */

    printf("max_output1 %f max_input1 %f max_input2 %f\n", max_output1, max_input1,
max_input2);
    printf("min_output1 %f min_input1 %f min_input2 %f\n", min_output1, min_input1, min_input2);
    getchar();

    if (fabs(max_input1) > fabs(min_input1))
    {
        input_slope1 = 1.0 / fabs(max_input1);
    }
    else
    {
        input_slope1 = 1.0 / fabs(min_input1);
    }

    if (fabs(max_input2) > fabs(min_input2))
    {
        input_slope2 = 1.0 / fabs(max_input2);
    }
    else
    {
        input_slope2 = 1.0 / fabs(min_input2);
    }

    if (fabs(max_output1) > fabs(min_output1))
    {
        output_slope1 = 1.0 / (fabs(max_output1 * 2.0));
    }
    else
    {
        output_slope1 = 1.0 / (int) ((fabs(min_output1 * 2.0)) + 1);
    }

```

```

for (index = 0; index < MAX_INPUT_INDEX; index++)
{
    array_input1[index] = array_input1[index] * input_slope1;
    array_input2[index] = array_input2[index] * input_slope2;
    array_output1[index] = 0.5 + (array_output1[index] * output_slope1);
}

max_input1 = array_input1[0];
max_input2 = array_input2[0];
max_output1 = array_output1[0];

min_input1 = array_input1[0];
min_input2 = array_input2[0];
min_output1 = array_output1[0];

for (index = 1; index < MAX_INPUT_INDEX; index++)
{
    if (max_input1 < array_input1[index])
    {
        max_input1 = array_input1[index];
    }

    if (min_input1 > array_input1[index])
    {
        min_input1 = array_input1[index];
    }

    if (max_input2 < array_input2[index])
    {
        max_input2 = array_input2[index];
    }

    if (min_input2 > array_input2[index])
    {
        min_input2 = array_input2[index];
    }

    if (max_output1 < array_output1[index])
    {
        max_output1 = array_output1[index];
    }
}

```

```

    }

    if (min_output1 > array_output1[index])
    {
        min_output1 = array_output1[index];
    }

}          /* End for index loop */

printf("max_output1 %f max_input1 %f max_input2 %f\n", max_output1, max_input1,
max_input2);
printf("min_output1 %f min_input1 %f min_input2 %f\n", min_output1, min_input1, min_input2);
getchar();

fclose(Error_in);
fclose(Fback_in);
fclose(Consigna_out);

if ((Max_input_value_fptr = fopen("maxinput.txt", "w")) == NULL)
{
    fprintf(stderr, "CAN NOT OPEN MAX INPUT VALUE FILE\n");
    return (FALSE);
}

if ((Error_fptr = fopen("error.txt", "w")) == NULL)
{
    fprintf(stderr, "CAN NOT OPEN ERROR FILE\n");
    return (FALSE);
}

if ((Fback_fptr = fopen("fback.txt", "w")) == NULL)
{
    fprintf(stderr, "CAN NOT OPEN FBACK FILE\n");
    return (FALSE);
}

if ((Consigna_fptr = fopen("consigna.txt", "w")) == NULL)
{
    fprintf(stderr, "CAN NOT OPEN CONSIGNA FILE\n");
    return (FALSE);
}

if ((Slope_fptr = fopen("slope.txt", "w")) == NULL)

```



```

{
    fprintf(stderr, "CAN NOT OPEN CONSIGNA FILE\n");
    return (FALSE);
}

```

```

fprintf(Max_input_value_fptr, "% 2.7e\n", max_input1);
fprintf(Max_input_value_fptr, "% 2.7e\n", min_input1);
fprintf(Max_input_value_fptr, "% 2.7e\n", max_input2);
fprintf(Max_input_value_fptr, "% 2.7e\n", min_input2);
fprintf(Max_input_value_fptr, "% 2.7e\n", max_output1);
fprintf(Max_input_value_fptr, "% 2.7e\n", min_output1);

```

```

for (index = 0; index < MAX_INPUT_INDEX; index++)
{

```

```

    number = array_input1[index];
    small = ((long) (number * 100000) * 10);
    big = (long) (number * 1000000);
    if ((big - small) >= 5)
    {
        number = (small += 10);
    }
    else
    {
        number = small;
    }

```

```

/* fprintf(Error_fptr, "% 2.7e\n", array_input1[index]); */
fprintf(Error_fptr, "% 2.7f\n", number / 1000000.0);

```

```

    number = array_input2[index];
    small = ((long) (number * 100000) * 10);
    big = (long) (number * 1000000);
    if ((big - small) >= 5)
    {
        number = (small += 10);
    }
    else
    {
        number = small;
    }

```

```

/* fprintf(Fback_fptr, "% 2.7e\n", array_input2[index]); */
fprintf(Fback_fptr, "% 2.7f\n", number / 1000000.0);

```

```

number = array_output1[index];

small = ((long) (number * 100000) * 10);
big = (long) (number * 1000000);

if ((big - small) >= 5)
{
    number = (small += 10);
}
else
{
    number = small;
}

/* fprintf(Consigna_fptr,"% 2.7e\n",array_output1[index]); */
fprintf(Consigna_fptr, "% 2.7f\n", number / 1000000.0);

}

fprintf(Slope_fptr, "% 2.7f\n", input_slope1);
fprintf(Slope_fptr, "% 2.7f\n", input_slope2);
fprintf(Slope_fptr, "% 2.7f\n", output_slope1);

fclose(Error_fptr);
fclose(Fback_fptr);
fclose(Consigna_fptr);
fclose(Max_input_value_fptr);
fclose(Slope_fptr);

return (TRUE);
}

```

```

#include <stdio.h>
#include "c:\borlandc\file\thesis.h"
#include "c:\borlandc\file\globalex.h"

int
read_input(void)
{
    int input_index;

    for (input_index = 0; input_index < MAX_INPUT_INDEX; input_index++)
    {

        fscanf(Error_fptr, "%f", &Inputx[0][input_index]);
        fscanf(Fback_fptr, "%f", &Inputx[1][input_index]);
        fscanf(Consigna_fptr, "%f", &Output[0][input_index]);

        /*
        * printf("Error_ptr %f\n",Inputx[0][input_index]); printf("Fback_ptr
        * %f\n",Inputx[1][input_index]); printf("Consigna_ptr
        * %f",Output[0][input_index]); getchar();
        */

    }

    fclose(Error_fptr);
    fclose(Fback_fptr);
    fclose(Consigna_fptr);

    if ((Slope_fptr = fopen("slope.txt", "r")) == NULL)
    {
        fprintf(stderr, "CAN NOT OPEN SLOPE FILE\n");
        return (FALSE);
    }

    fscanf(Slope_fptr, "%f", &Input_slope1);

    fscanf(Slope_fptr, "%f", &Input_slope2);

    fscanf(Slope_fptr, "%f", &Output_slope1);

    fclose(Slope_fptr);

    if ((Max_data_fptr = fopen("maxinput.txt", "r")) == NULL)

```

```

{
    fprintf(stderr, "CAN NOT OPEN MAX INPUT FILE\n");
    return (FALSE);
}

for (input_index = 0; input_index < ((2 * INPUT_LINGUI_NO) + (2 * OUTPUT_LINGUI_NO));
input_index++)
{
    fscanf(Max_data_fptr, "%f", &Max_input1);
    fscanf(Max_data_fptr, "%f", &Min_input1);
    fscanf(Max_data_fptr, "%f", &Max_input2);
    fscanf(Max_data_fptr, "%f", &Min_input2);
    fscanf(Max_data_fptr, "%f", &Max_output1);
    fscanf(Max_data_fptr, "%f", &Min_output1);
}

printf("Max_input1 %f\n", Max_input1);
printf("Min_input1 %f\n", Min_input1);
printf("Max_input2 %f\n", Max_input2);
printf("Min_input2 %f\n", Min_input2);
printf("Max_output1 %f\n", Max_output1);
printf("Min_output1 %f\n", Min_output1);

fclose(Max_data_fptr);
getchar();

return (TRUE);
}

```

REFERENCES

- [1] Anderson, W. Charles. "Learning to Control an Inverted Pendulum Using Neural Networks," IEEE Control Systems Magazine, April 1989, pp.31-37.
- [2] Barto, G. Andrew., Sutton, S. Richard., and Anderson, W. Charles. "Neuronlike adaptive Elements That Can Solve Difficult Learning Control Problems," IEEE Transactions on Systems, Man, and Cybernetics, 13:5, September/ October 1983, pp. 835-846.
- [3] Braae, M., and Rutherford, D. A. "Fuzzy relations in a control setting," Kybernetes, 7:3, 1978, pp. 185-188.
- [4] Chuen-Tsai Sun. "Rule-Base Structure Identification in an Adaptive-Network-based Fuzzy Inference System," IEEE Transactions on Fuzzy System, 2:1, February 1994, pp. 64-73.
- [5] Feldman, J. A., and Ballard, D. H. "Connectionist Models and Their Properties," Cognitive Science, 6, 1982, pp. 205-254.
- [6] Franklin, A. Judy., Sutton S. Richard., and Anderson, W. Charles. Connectionist Learning Control at GTE Laboratories, Proceeding of the SPIE 1989 Symposium on Advances in Intelligent Robotics Systems. Philadelphia, Pennsylvania, November 1989.
- [7] Gallant, S. I.. "Connectionist Expert Systems," Commun. ACM, 31:2, Feb 1988, pp. 152-169.
- [8] Gold, B. Hopfield Model Applied to Vowel and Consonant Discrimination, MIT Lincoln Laboratory Technical Report, TR-747, AD-A169742, June 1986.
- [9] Groosberg, S. The adaptive Brain I: Cognition, Learning, Reinforcement, and Rhythm, and The Adaptive Brain II: Vision, Speech, Language, and Motor control, Elsevier/Norh-Holland, Amsterdam 1986.
- [10] Hebb, D. O. The Organization of Behavior, John Wiley & Sons, New York 1949.
- [11] Holpfield, J. J. "Neurons with Graded Response Have Collective Computational Properties Like those of Two-state Neurons," Proc. Natl. Acad. Sci. USA, 81, May 1984, pp. 3088-3092.
- [12] Hopfield, J. J. and Tank, D. W. "Computing with Neural Circuits: A Model," Science, 233, August 1986, pp. 625-633.
- [13] Kohonen, T., Masisara, K., and Saramaki, T. Phonotopic Maps - Insightful Representation of Phonological Features for Speech Representation, Proceedings IEEE 7th Inter. Conf. on Pattern Recognition, Montreal, Canada, 1984.
- [14] Kohonen, T. Self-Organization and Associative Memory, Springer-Verlag, Ber-

lin 1984.

- [15] Langari, Reza., and Berenji, R. Hamid. "Fuzzy Logic in Control engineering," Handbook of Intelligent Control, D.A. White and D.A. Sofge (Eds), Van Nostrand Reinhold, New York, 1992, pp. 93-140.
- [16] Layne, J. R., Passino, M. Kevin., and Yurkovich, Yurkovich. "Fuzzy Learning Control for antiskid Braking Systems," IEEE Transactions Control System Technology, 1:2, June 1993, pp. 122-129.
- [17] Layne, J. R., and Passino, M. Kevin. "Fuzzy Model Reference Learning Control for Cargo Ship Steering," IEEE Control Systems Magazine, December 1993, pp. 23-34.
- [18] Lin, Chin-Teng., and Lee, C. S. George. "Neural-Network-Based Fuzzy Logic Control and Decision Systems," IEEE Transactions on Computers, 40:12, December 1991, pp. 1320-1336.
- [19] Lin, Chin-Teng., and Lee, C. S. George. "Reinforcement Structure/Parameter Learning for Neural-Network-Based Fuzzy Logic Control Systems," IEEE Transactions on Fuzzy Systems, 2:1, February 1994, pp. 46-63.
- [20] Linkens, D. A., and Hasnain, S.B. "Self-Organising Fuzzy Logic Control and Application to Muscle Relaxant Anaesthesia", IEE Proceedings-D, 138:3, May 1991, pp. 274-284.
- [21] Lippmann, R. P., Gold, B., and Malpass, M. L. A Comparison of Hamming and Hopfield Neural Nets for Pattern Classification, MIT Lincoln Laboratory Technical Report, TR-769.
- [22] Lippmann, R. P. "An Introduction to Computing with Neural Nets," IEEE ASSP Magazine, April 1987, pp. 4-22.
- [23] McCulloch, W. S., and Pitts, W. "A Logical Calculus of the Ideas Imminent in Nervous Activity," Bulletin of Mathematical Biophysics, 5, 1943, pp. 115-133.
- [24] Posch, T. E. Models of the Generation and Processing of Signals by Nerve Cells: A Categorically Indexed Abridged Bibliography, USCEE Report 290, August 1968.
- [25] Procyk, T. J., and Mamdani, E. H. "A linguistic Self-Organising Process Controller," Automatica, 1, 1979, pp. 15-30.
- [26] Rosenblatt, R. "Principles of Neurodynamics," Spartan Books, New York, 1959.
- [27] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. Learning Internal Representations by Error Propagation, in D. E. Rumelhart & J. L. McClelland (Eds.), Parallel Distributed Processing: Explorations in the Microstructure of Cognitions, Vol.1, MIT Press (1986).

- [28] Touret, D. S., and Hinton, G. E. A Distributed Connectionist Production System. CMU-CS-86-172, Technical Report Department of Computer Science, Carnegie Mellon Univ., 1986.
- [29] Tzes, Anthony., and Yurkovich, Stephen. "An Adaptive Input Shaping Control Scheme for Vibration Suppression in Slewing Flexible Structures," IEEE Transactions Control Systems Technology, 1:2, June 1993, pp 114-121.
- [30] Widrow, B., and Hoff, M. E. "Adaptive switching Circuits," 1960 IRE WESCON Convention. Record, Part 4, August 1960, pp. 96-104.
- [31] Zadeh, L. A. "Fuzzy sets," Information and Control, 8, 1965, pp. 28-44.
- [32] Zhang, B. S., and Edmunds, J. M. "Self-organizing fuzzy logic controller," IEE Proceedings-D, 139:5, September 1992, pp 460-464.